ISBN4-88554-153-X-C3055 ¥980E

定価980円

HOW TO マシン語
Z80/6809
マスター編

電波新聞社

★★絵で見るマシン語プログラム★★

HOW TO マシン語

Z80/6809マスター編

絵/秋山早苗
監修/大沢正道



ISBN4-88554-153-X-C3055 ¥980E

定価980円

HOW TO マシン語 Z80/6809マスター編

電波新聞社

★★絵で見るマシン語プログラム★★

HOW TO マシン語

絵/秋山早苗
監修/大沢正道

Z80/6809マスター編



★★絵で見るマシン語プログラム★★

HOW TO マシン語

Z80/6809マスター編



はじめに

パソコンのシステム本体内には，ＢＡＳＩＣと呼ばれるインタプリタ形式の高級言語が装備されています。このＢＡＳＩＣは，初心者の方でも理解しやすくプログラム作成が容易な自然言語で，このため広く普及しているのは周知のとおりです。ところで一方のマシン語は，ＢＡＳＩＣとは桁違いの処理速度を秘めているにもかかわらずあまり知られていません。その理由は，マシン語がプログラムを記述する言語というより，意味不明な数値を羅列する言語だといっても過言ではないのです。そこで本書は，このようなマシン語という奇々怪々な言語を習得しようと志された勇気ある技術者やホビーストのみなさんを対象にし，従来に無いコミック形式で超初級マシン語入門書を編集しました。したがって，これからマシン語を勉強されるマシン語新入生の方にとっては，このうえない一冊となりうることうけあいです。もちろんマシン語留年生諸君も，本書を使って挫折したところから再びチャレンジされれば大丈夫でしょう。

さて，本書は，内容を大きく３部に分け，つぎのように構成しています。

第一部　マシン語の基礎（How To テクニック編）
第二部　Ｚ80ＣＰＵマシン語入門
第三部　6809ＣＰＵマシン語入門

なんと，マシン語の基礎はもちろんのこと，Ｚ80／6809という二つものＣＰＵのマシン語が書かれているという贅沢な入門書なのです。

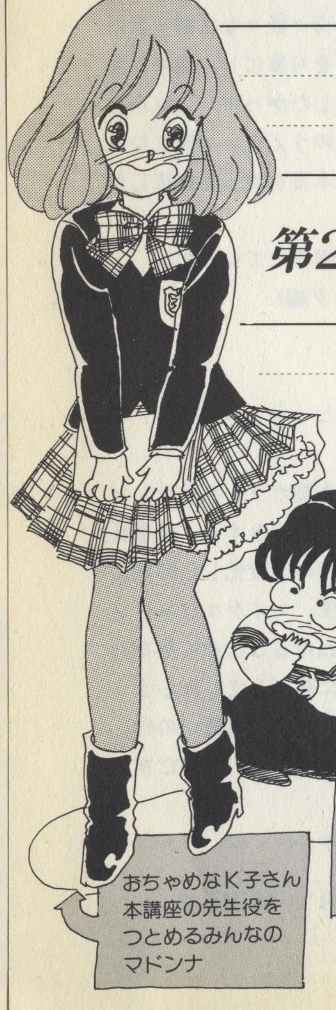
最後に，日本シリーズ真っ最中にもかかわらずデスクに小型カラー液晶テレビを忍ばせることなく編集作業をしていただいた担当さんをはじめとする，編集長，ＫＥＩＫＯさんの月刊マイコン編集部の皆様，卒業論文制作の大変忙しい時期にもかかわらず原稿を書いていただいた漫画家の秋山早苗先生，マイケルジャクソンのジャパンツアーＴＶ放映録画したさにビデオデッキをねだることなく毎夜お弁当を運んでくれた妹，担当さんとの打ち合せのため早朝に起こしてくれるよう頼んでいたにもかかわらず睡眠を妨げず熟睡させてくれるやさしい気遣いの母，自分を「将来性抜群」と無理やり電気技術者の道に進めてくれた，いまは亡き父に……

以上，いろいろお世話になった皆様に感謝の意を表します。

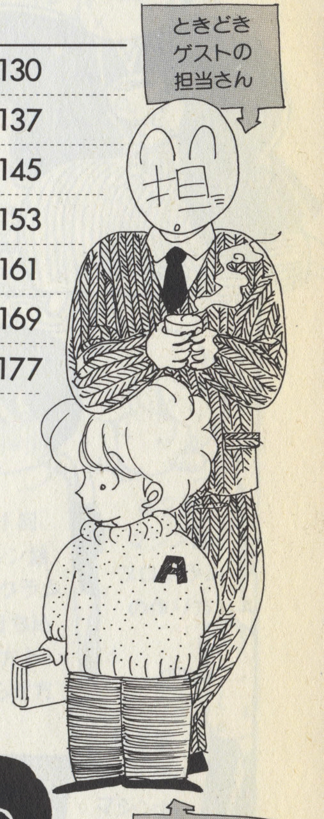
1987年11月　　大沢正道

絵で見るマシン語プログラム

How To マシン語 Z80/6809 マスター編









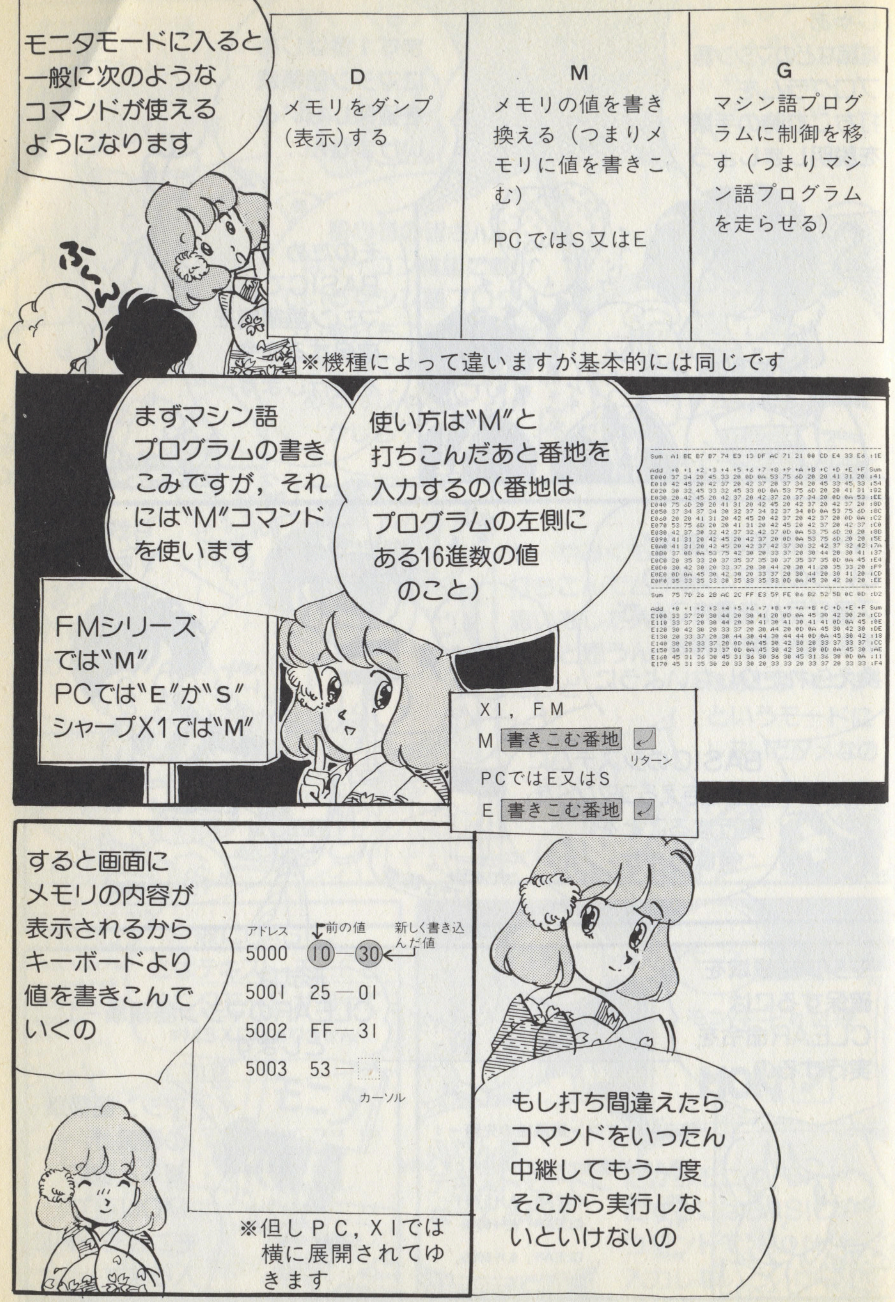
第1部 マシン語プログラム基礎編

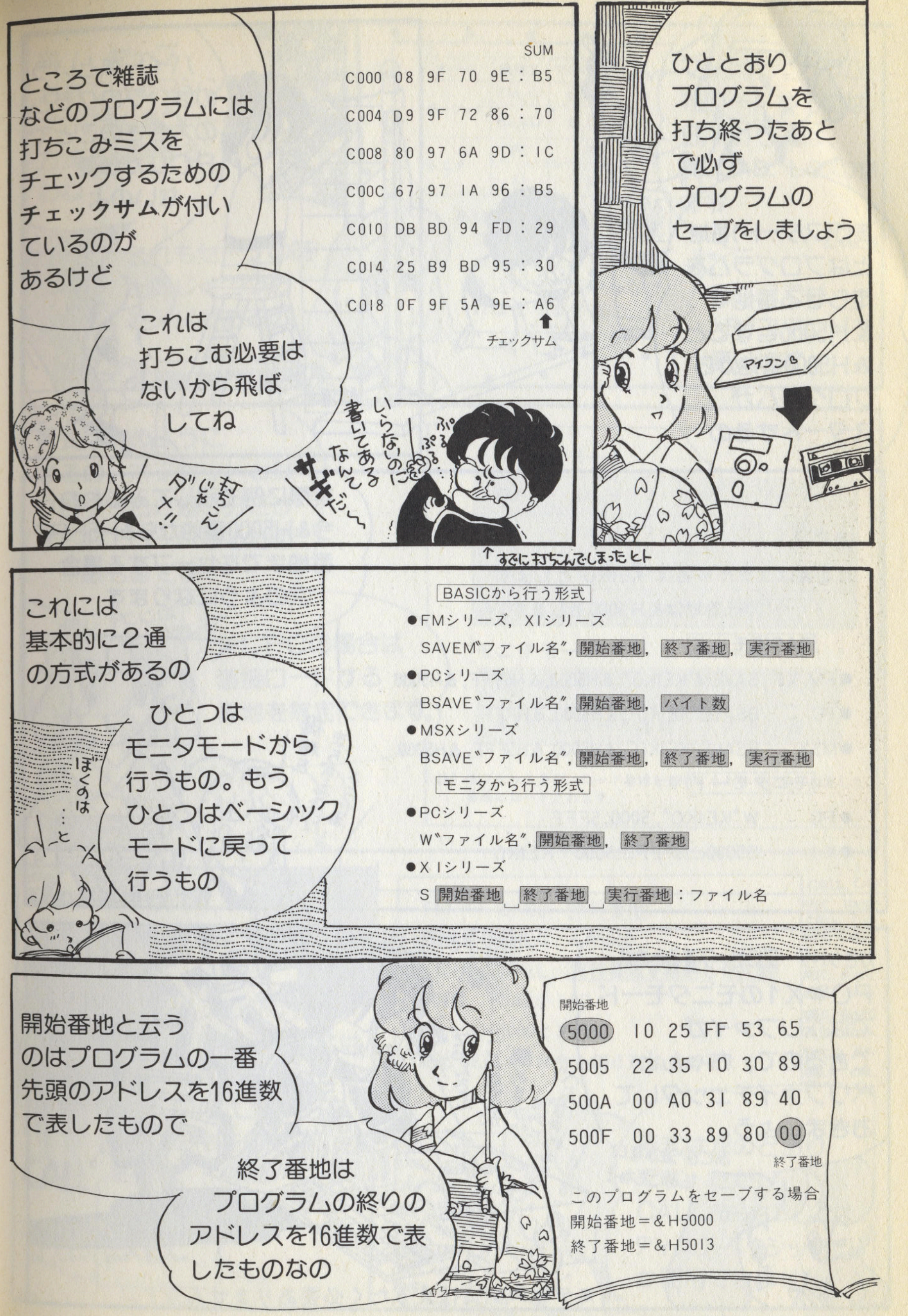
1-1 マシン語プログラムの打ち込み方と走らせ方

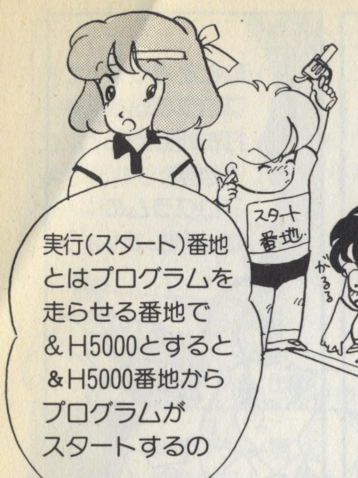


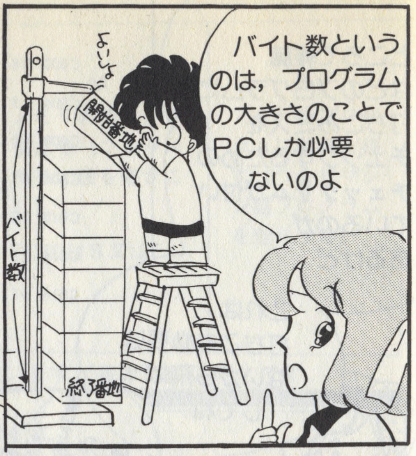

まずはマシン語プログラムの打ちこみ方と走らせ方について勉強します
8000 8E 80 13 CE 01 F9 C6 0A
8008 BD 85 97 8E 80 26 C6 0A
8010 A0 A0 A0 A0 A0 A0 06 80
8018 30 80 5A 01 80 71 80 77
8020 7E 85 97 06 80 10 80 57
8028 09 80 1D 80 77 A0 A0 A0
初心者の皆さんはマイコン雑誌に載っているマシン語プログラムを打ちこんで走らせようとして悩んだ事があるんじゃないかしら？
初体験
問題はまずどうやってマシン語を打ちこむか！
殿中でござる！
お離しくだされ武士の情けでござる〜
ずっぱーん
や…やみくもに打ちこんでムダな思いをした人もいると思うんだけど
実はマシン語を打ちこむためには“モニタ”というモードにしないとダメなの
一般にBASICにはテキスト・モードとモニタモードがあって
※MSXにはありません
テキスト
モニタ
テキスト
MON⏎
モニタ
モニタモードの切換
MON⏎
ふだんはテキストモードになってるわけ
だからモニタ・モードにするにはBASICのコマンドで“MON”と入力しないといけないの


じゃあ雑誌などのマシン語プログラムを打ちこむ時の手順を説明しましょう
パチパチ
まず1番はじめにマシン語領域を確保しないといけません
そのためBASICでマシン語領域を確保する命令を実行します
“マシン語領域の確保”というのはマシン語のプログラムを書きこんだ所を消されたり書き換えられたりしないように
ユーザー領域
BASIC
BASICのシステムに忠告を与えるコマンドを実行することを意味します
プログラム システム 領域
マシン語領域
マシン語領域を確保するにはCLEAR命令を実行するの
形式はCLEARのマシン語領域−1⏎としします
CLEAR, マシン語領域の先頭−1
例 &H6000番地からマシン語領域とする場合
PC, FM　CLEAR, &H5FFF
X1　CLEAR &H6000
MSX　CLEAR, &H6000
注1 X1は“,”はいりません X1, MSXでは，パラメータは−1する必要はありません
注2 この命令をサブルーチン内で使ってはいけません。
マシン語領域の確保が終ると次にMON⏎でモニタモードに入ります


モニタモードに入ると一般に次のようなコマンドが使えるようになります
ふらん
D
メモリをダンプ(表示)する
M
メモリの値を書き換える(つまりメモリに値を書きこむ)
PCではS又はE
G
マシン語プログラムに制御を移す(つまりマシン語プログラムを走らせる)
※機種によって違いますが基本的には同じです
まずマシン語プログラムの書きこみですが，それには"M"コマンドを使います
使い方は"M"と打ちこんだあと番地を入力するの(番地はプログラムの左側にある16進数の値のこと)
FMシリーズでは"M"
PCでは"E"か"S"
シャープX1では"M"
X1，FM
M 書きこむ番地 ↵
リターン
PCではE又はS
E 書きこむ番地 ↵
すると画面にメモリの内容が表示されるからキーボードより値を書きこんでいくの
アドレス
前の値
新しく書き込んだ値
5000 10—30
5001 25—01
5002 FF—31
5003 53—
カーソル
※但しPC，X1では横に展開されてゆきます
もし打ち間違えたらコマンドをいったん中継してもう一度そこから実行しないといけないの


ところで雑誌などのプログラムには打ちこみミスをチェックするためのチェックサムが付いているのがあるけど
SUM
C000 08 9F 70 9E : B5
C004 D9 9F 72 86 : 70
C008 80 97 6A 9D : 1C
C00C 67 97 1A 96 : B5
C010 DB BD 94 FD : 29
C014 25 B9 BD 95 : 30
C018 0F 9F 5A 9E : A6
チェックサム
これは打ちこむ必要はないから飛ばしてね
打ちこんじゃダメ!!
いらないのに書いてあるなんてサギだ〜
ぷるぷる
↑すでに打ちこんでしまったヒト
ひととおりプログラムを打ち終ったあとで必ずプログラムのセーブをしましょう
マイコン
これには基本的に2通の方式があるの
ひとつはモニタモードから行うもの。もうひとつはベーシックモードに戻って行うもの
ぼくのは…と
BASICから行う形式
●FMシリーズ，X1シリーズ
SAVEM"ファイル名",開始番地, 終了番地, 実行番地
●PCシリーズ
BSAVE"ファイル名",開始番地, バイト数
●MSXシリーズ
BSAVE"ファイル名",開始番地, 終了番地, 実行番地
モニタから行う形式
●PCシリーズ
W"ファイル名",開始番地, 終了番地
●X1シリーズ
S 開始番地 終了番地 実行番地：ファイル名
開始番地と云うのはプログラムの一番先頭のアドレスを16進数で表したもので
終了番地はプログラムの終りのアドレスを16進数で表したものなの
開始番地
5000 10 25 FF 53 65
5005 22 35 10 30 89
500A 00 A0 31 89 40
500F 00 33 89 80 00
終了番地
このプログラムをセーブする場合
開始番地＝&H5000
終了番地＝&H5013





例

たとえばファイル名は"KEIKO"とします

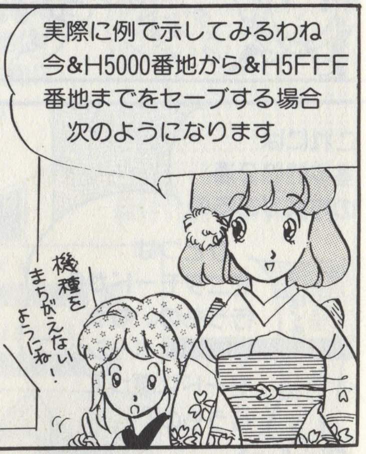
（実行番地は＆H5000とします）

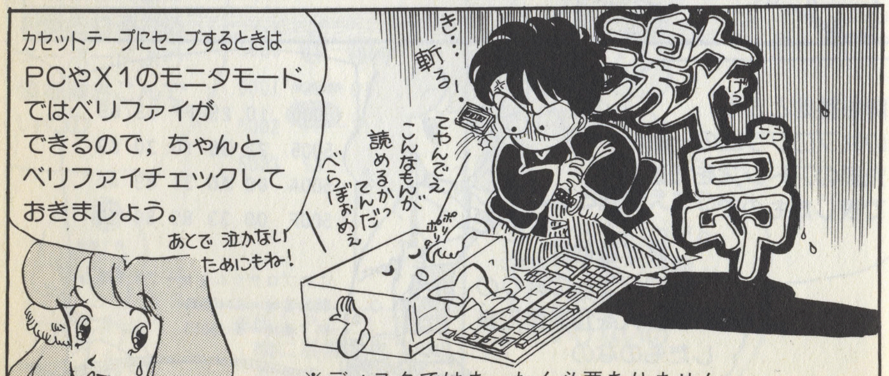
BASICモード

- FM, X1　SAVEM"KEIKO", &H5000, &H5FFF, &H5000
- PC　BSAVE"KEIKO", &H5000, &HFFF
- MSX　BSAVE"KEIKO", &H5000, &H5FFF, &H5000

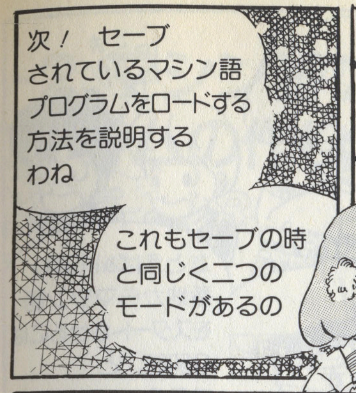
モニタモード

- PC　W"KEIKO", 5000, 5FFF
- X1　S5000␣5FFF␣5000：KEIKO



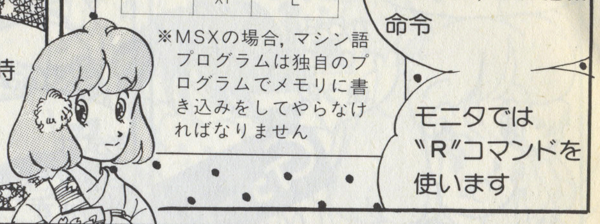


※ディスクではまったく必要ありません



モード	機種名	マシン語モード
ベーシック	FM, X1	LOADM
	PC, MSX	BLOAD
モニタ	PC	R
	X1	L

※MSXの場合，マシン語プログラムは独自のプログラムでメモリに書き込みをしてやらなければなりません





BASICモード

FM	LOADM"ファイル", オフセット
X1	LOADM"ファイル", ロード番地
PC	BLOAD"ファイル", ロード番地
MSX	BLOAD"ファイル", オフセット

モニタモード

PC	R
X1	Lロード番地：ファイル名

注）オフセット及び，ロード番地を省略するとセーブ時に指定した番地にロードされます。



FM	EXEC	実行番地
（例）	EXEC	&H5000
X1, PC	CALL	実行番地を格納した変数
（例）	CALL	AD

AD＝&HC000とすれば

あとDEFUSR命令を使う形式もあります





 ＊誤動作するという意味です

1-2 マシン語プログラムの作り方




どど…ど
どーしたの突然！
熱でもあるんじゃ
ないのっ？
重症の
正月ボケ
かな？
ぱっ
ちがわい
じた
ばた
まあ
すてき
マシン語を
マスター
↓
すごい
↓
クラスの
人気者
なーんだ
くだらん！
ガールフレンド
彼氏の
いなり
ねたみ
動機は不純だけど
まあいいわ マシン語
プログラムの作り方を
教えてあげるね
う～ん
読めん
ふんふん
なるほど
ピラ
ピラ
マシン語
そもそもマシン語は
コンピュータの理解する
唯一の命令 つまり
“ことば”なの
それらはBASIC
とはちがって２進法
で表わされているの
ただこのままじゃ，取扱いに
困るから16進数に変換した
ものを使うの
だからマシン語
プログラムとはこれらを
使って作られたプログラム
のことなのよ
ほえ～
㊎Aレンジに１をたす命令（Z80）
２進法 00111100
↓
00111100
3 C
↓
16進に
変換
3C
マシン語プログラムとは
こういった命令で作った
プログラムのことです


じゃK子さん
マシン語でプログラム
を作ろうと思ったら
この暗号みたいな
のを覚えないと
だめなの？
もちろん！
これらは
正確には
インストラクション
・コードって
いうのよ
なんと
げげっ
Z80
≠
6809
げ
わかる？
ワニのうでたてふせよりムリだ
しかも
インストラクション・
コードはCPUによって
全部違っていて
たとえばFM-7/77の
6809とPC88，X１の
Z80だと全く別の
インストラクション・コードを
使うっていうわけ
プログラムのつくり方
はじまりはじまり
じゃそろそろ
プログラムの作り方を
説明しようか
まずCPUには
インストラクション・コードで
表わされる命令があると
いうのは説明したけど
それじゃ実際に
6809とZ80の
インストラクション・コード
を見てみることに
しましょ
軟派な
心構えじゃ
だめよ
「まあ♡〇〇さん．すてき♡」
GF
WIFE
LOVER
硬派です
硬派！
ちょっと
わかりにくいから
覚悟してね
真剣なまなざし．

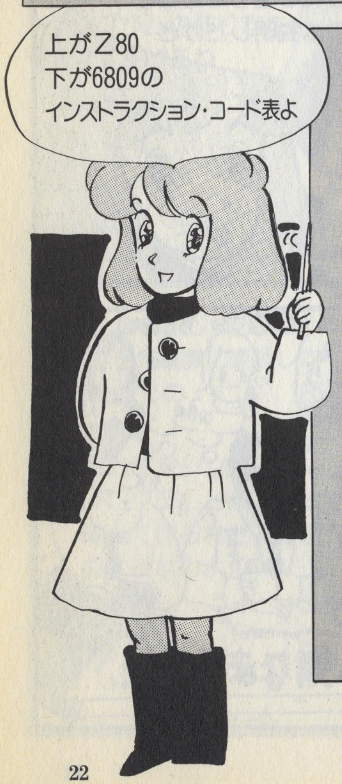
Z80 インストラクション・コード表(30〜3F)＊

命令	内容	
30	キャリーフラグが"1"でないなら分岐	JR NC, PP
31	SPレジスタに値をロードする	LD SP, nn
32	Aレジスタをメモリにセーブする	LD (nn), A
33	SP＝SP＋1	INC SP
34	(HL)＝(HL)＋1	INC (HL)
35	(HL)＝(HL)－1	DEC (HL)
36	nをHLレジスタで指されるメモリにセーブする	LD (HL), n
37	キャリーフラグをセットする	SCF
38	キャリーフラグが"1"なら分岐する	JP C, PP
39	HL＝HL＋SP	ADD HL, SP
3A	Aレジスタにメモリの値をロードする	LD A, (nn)
3B	SP＝SP－1	DEC SP
3C	A＝A＋1	INC A
3D	A＝A－1	DEC A
3E	Aレジスタに値をロードする	LD A, n
3F	キャリーフラグのビットを反転する	CCF

PPはレラティブ，アドレッシングモードにおける変位値nnは16ビットの値，nは8ビットの値

(nn)はnnで表わされるメモリアドレス



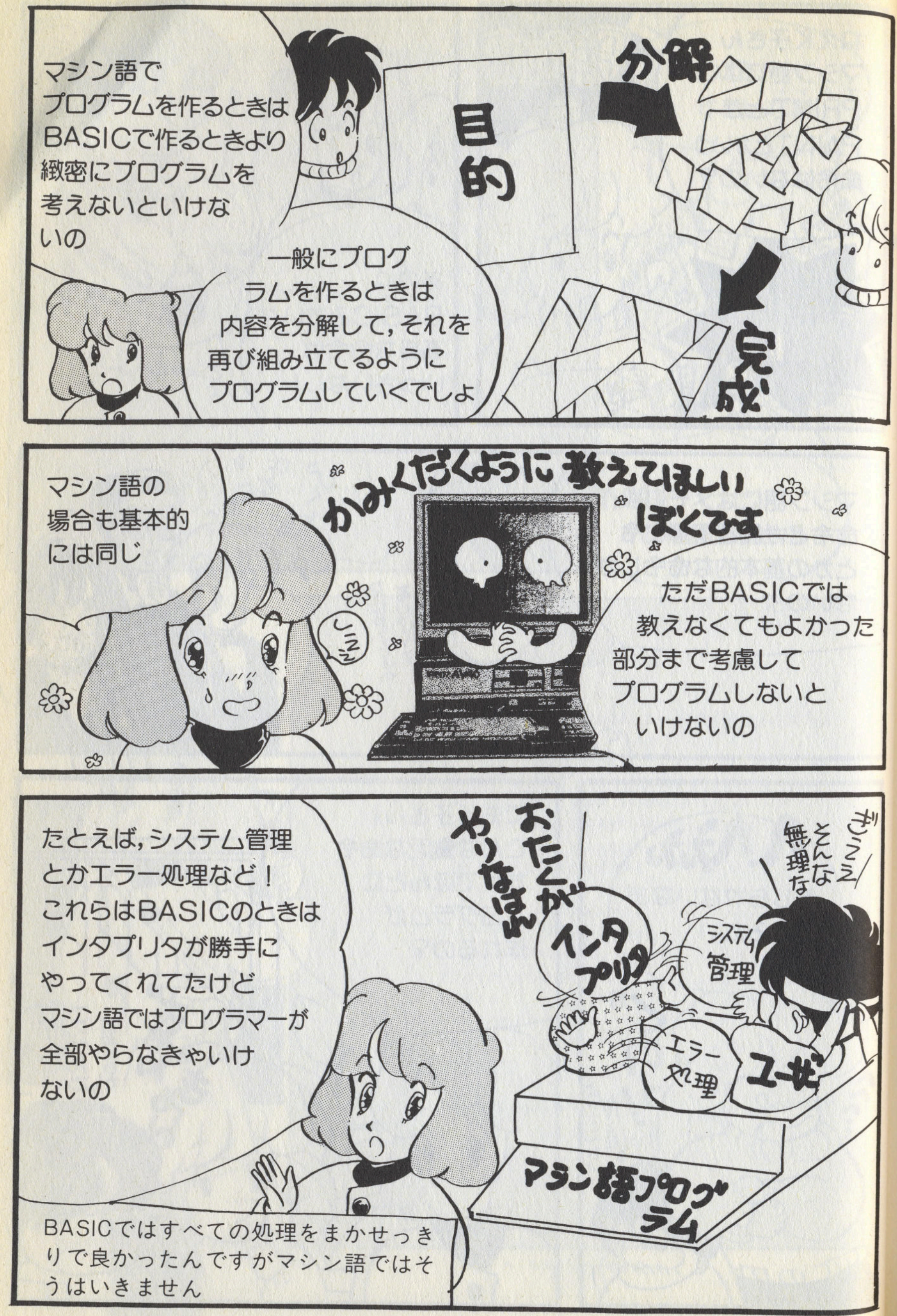


6809のインストラクション・コード表(表30〜3F)＊

命令	内容	
30	Xレジスタに実効アドレスをロードする	LEAX
31	Yレジスタに実効アドレスをロードする	LEAY
32	システムSPに実効アドレスをロードする	LEAS
33	ユーザーSPに実効アドレスをロードする	LEAU
34	システムSPにレジスタを退避する	PSHS
35	システムSPからレジスタを復帰する	PULS
36	ユーザーSPにレジスタを退避する	PSHU
37	ユーザーSPからレジスタを復帰する	PULU
38	――――	――
39	サブルーチンから復帰する	RTS
3A	X＝X＋B	ABX
3B	割り込み先から復帰する	RTI
3C	割り込みを待つ(CCRがクリアされる)	CWAI
3D	D＝A×B	MUL
3E	――――	――
3F	ソフトウェア割り込み	SWI

＊これはほんの一部です





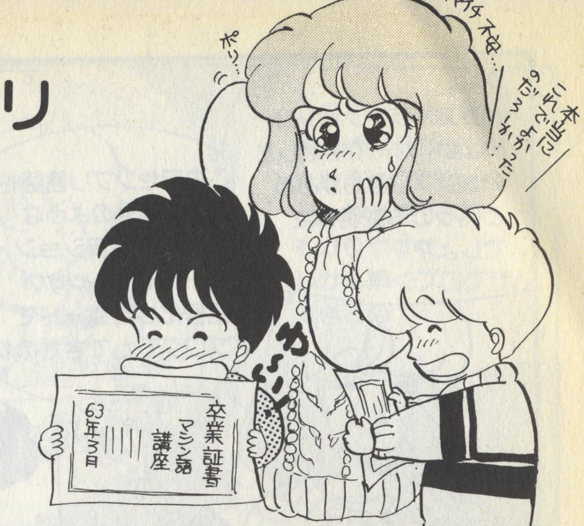
マシン語で
プログラムを作るときは
BASICで作るときより
緻密にプログラムを
考えないといけな
いの
目的
分解
完成
一般にプログ
ラムを作るときは
内容を分解して，それを
再び組み立てるように
プログラムしていくでしょ
マシン語の
場合も基本的
には同じ
かみくだくように教えてほしいぼくです
ただBASICでは
教えなくてもよかった
部分まで考慮して
プログラムしないと
いけないの
たとえば，システム管理
とかエラー処理など！
これらはBASICのときは
インタプリタが勝手に
やってくれてたけど
マシン語ではプログラマーが
全部やらなきゃいけ
ないの
おたくがやりなはれ
そんな無理な
インタプリタ
システム管理
エラー処理
ユーザ
マシン語プログラム
BASICではすべての処理をまかせっき
りで良かったんですがマシン語ではそ
うはいきません

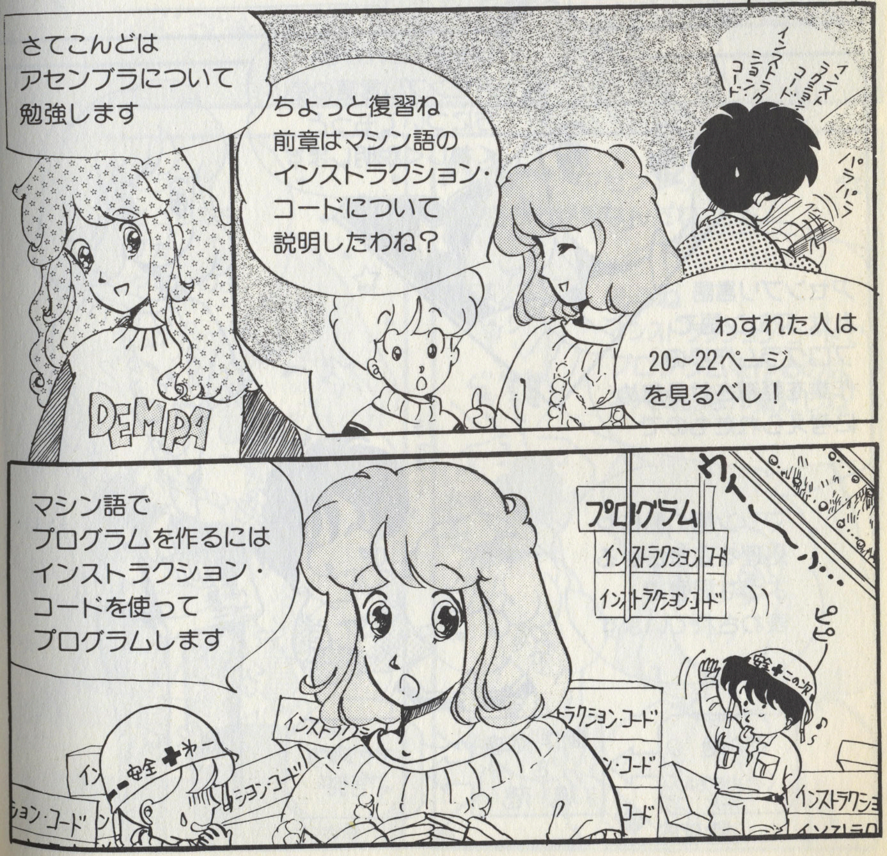


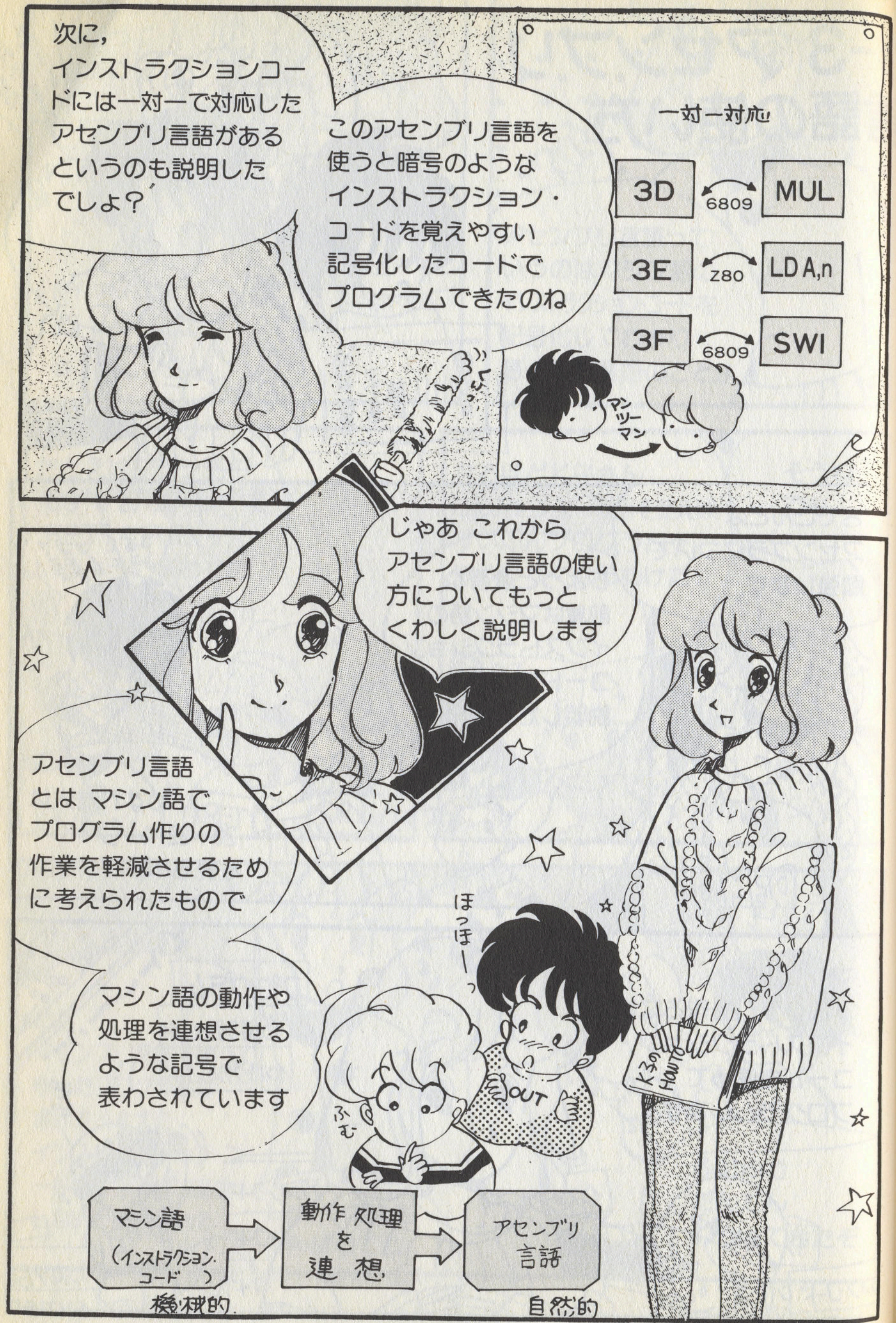
マシン語作りの基本
として
①どんな処理をさせる
か考える
②実際に手順を考える
③手順どおりにマシン
語を組み上げていく
②
手順を考える
B=B-1
IFB=0
NO
YES
C=0
RET
①
目的
③
インストラクションコード表
というように
進めていくの
なるほど
こうしてさせたいことを
砕きながら少しずつ
プログラムしていけば
いいのか
よいしょ
プログラム構築
ところでK子
さん
さっきのインストラク
ション・コード表(P.22)
の右側になら
んでるのは
なあに？
あ，あれ？
あれは
アセンブリ
言語よ

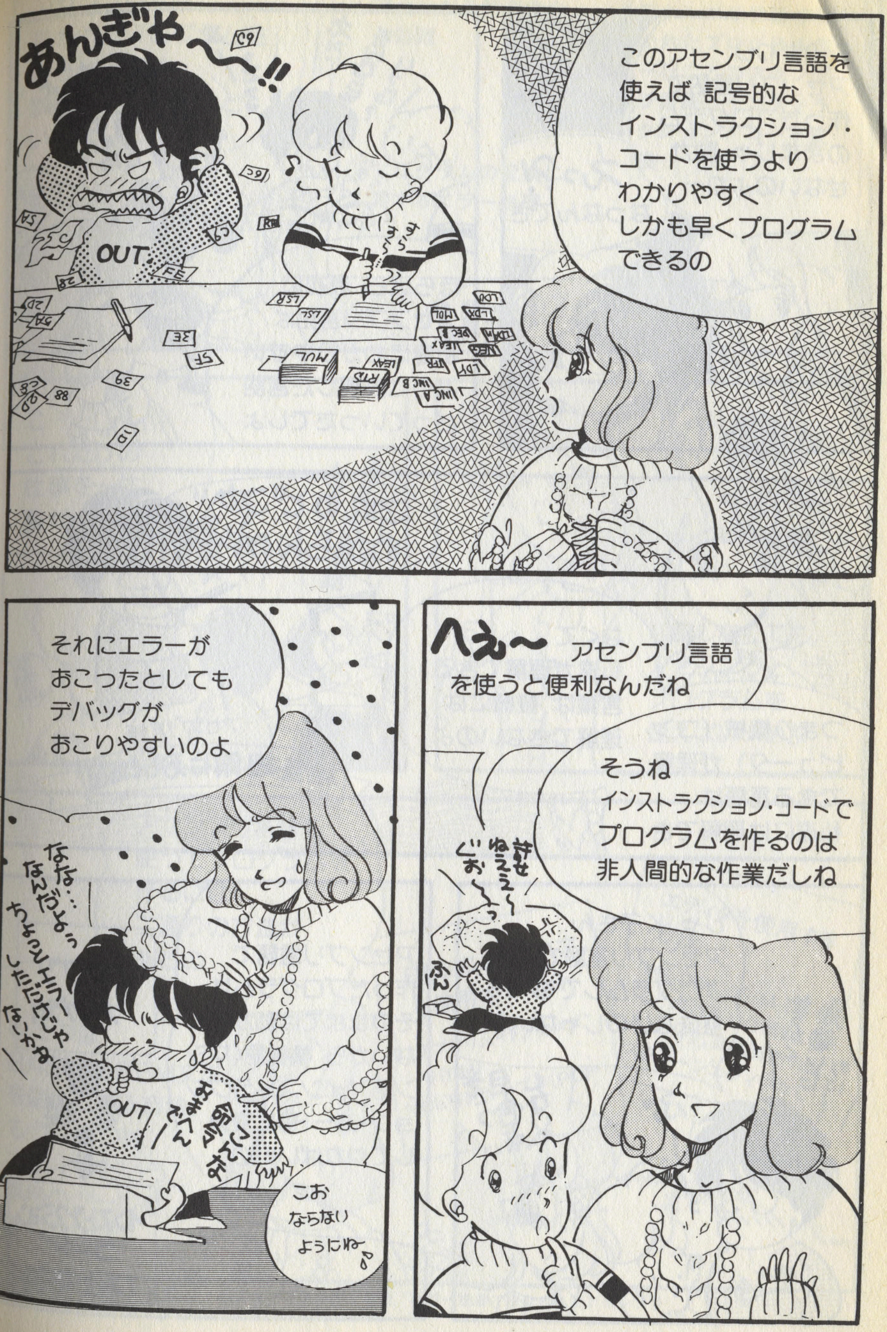


1-3 アセンブリ言語の使い方

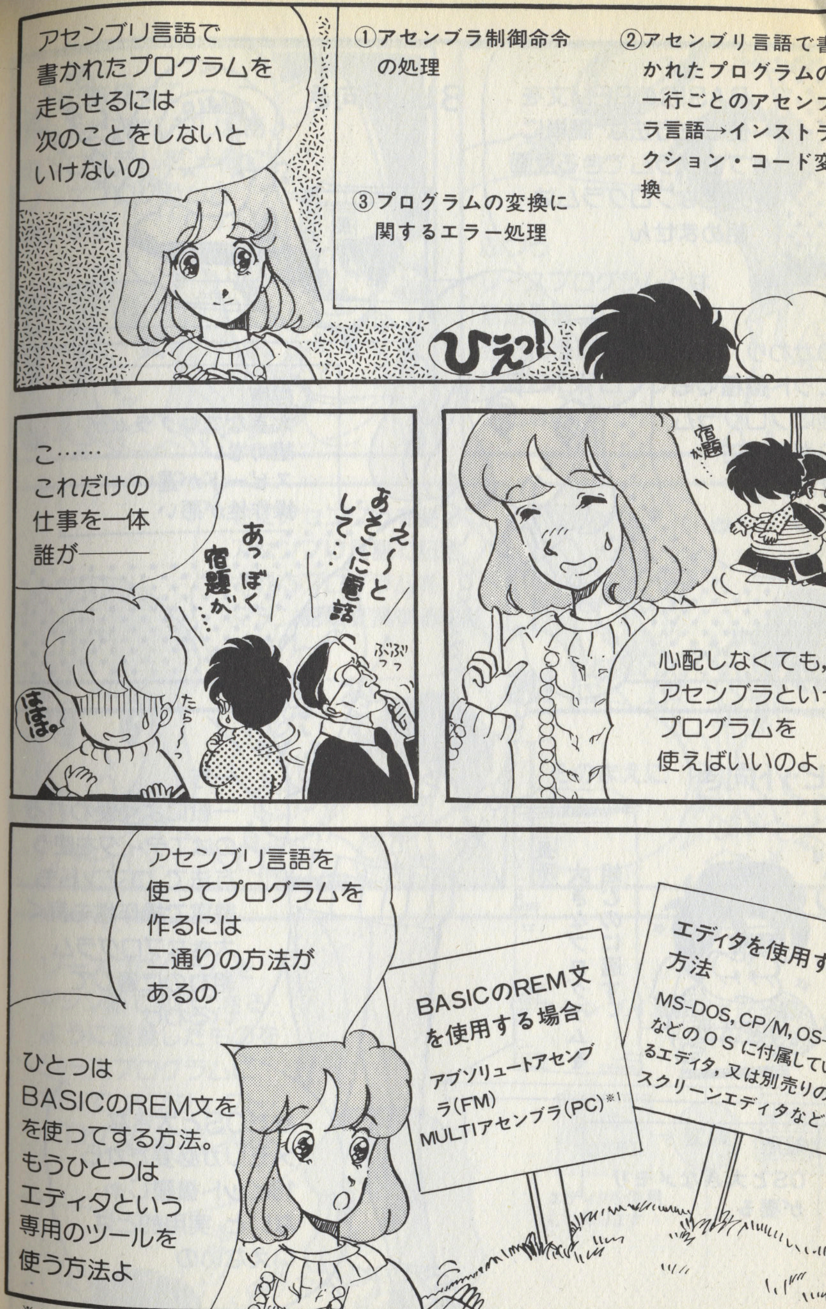




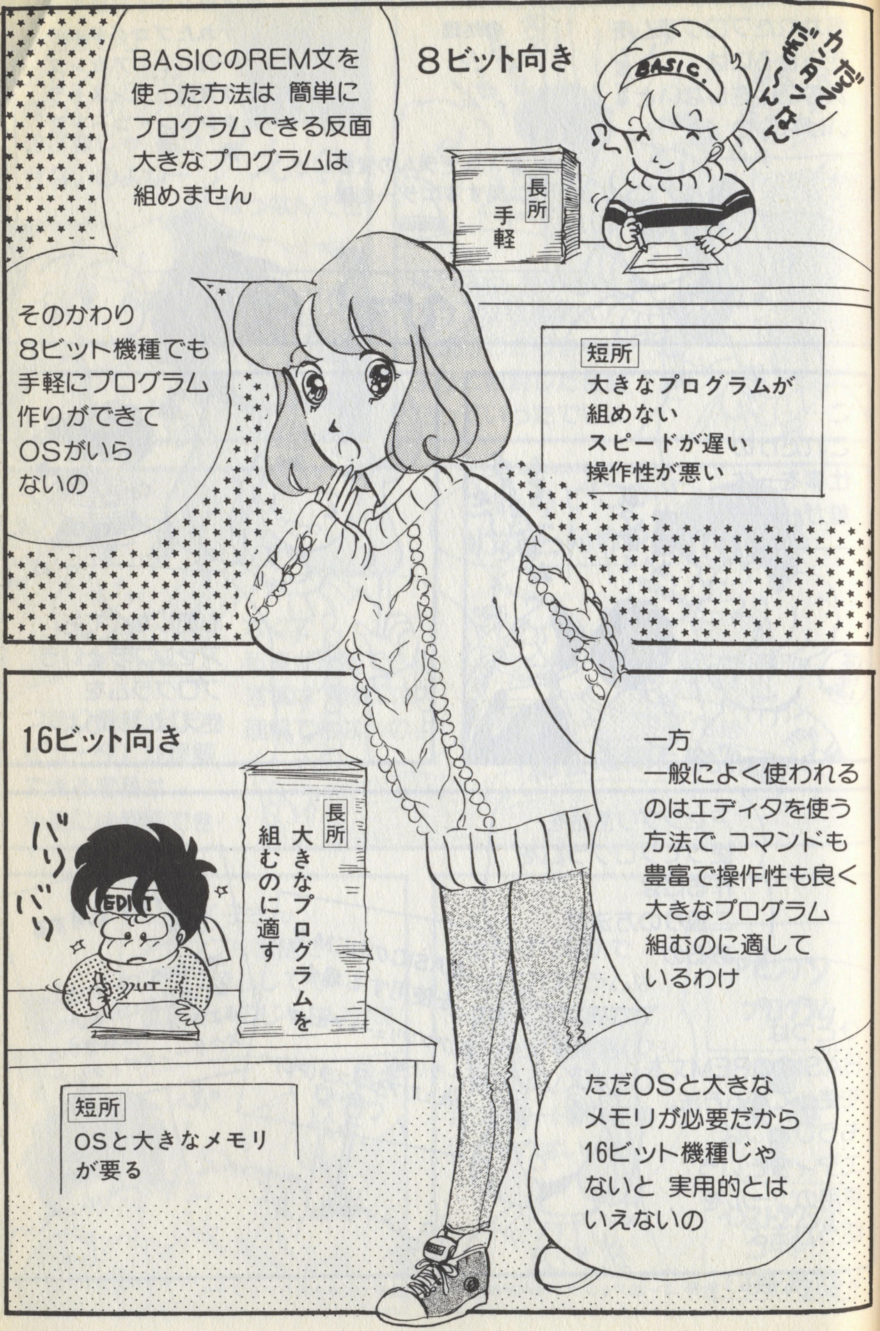

次に，
インストラクションコードには一対一で対応したアセンブリ言語があるというのも説明したでしょ？
このアセンブリ言語を使うと暗号のようなインストラクション・コードを覚えやすい記号化したコードでプログラムできたのね
一対一対応
3D 6809 MUL
3E Z80 LD A,n
3F 6809 SWI
じゃあ これからアセンブリ言語の使い方についてもっとくわしく説明します
アセンブリ言語とは マシン語でプログラム作りの作業を軽減させるために考えられたもので
マシン語の動作や処理を連想させるような記号で表わされています
ほっほー
ふむ
マシン語（インストラクション・コード）
機械的
動作 処理 を 連想
アセンブリ言語
自然的


あんぎゃ〜〜!!
このアセンブリ言語を使えば 記号的なインストラクション・コードを使うよりわかりやすく しかも早くプログラムできるの
OUT
へぇ〜 アセンブリ言語を使うと便利なんだね
そうね インストラクション・コードでプログラムを作るのは非人間的な作業だしね
それにエラーがおこったとしてもデバッグがおこりやすいのよ
なな…なんだよぅ ちょっとエラーしただけじゃないかよ
こお ならないようにね

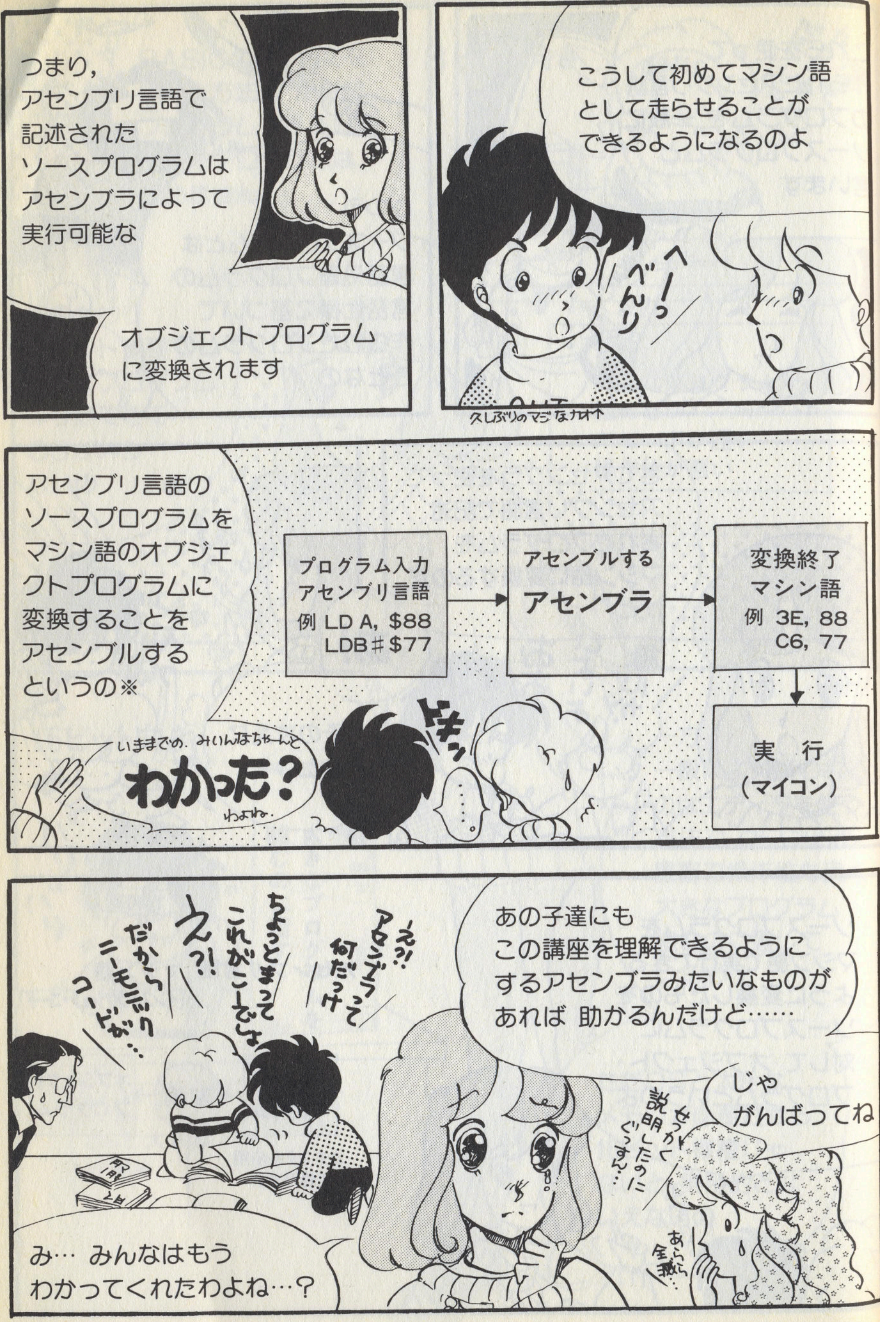




※1　月刊マイコン別冊　Ｚ－80アセンブラの内部構造／塚越一雄著で紹介されたアセンブラ


BASICのREM文を使った方法は 簡単にプログラムできる反面 大きなプログラムは組めません
8ビット向き
BASIC
長所
手軽
そのかわり 8ビット機種でも手軽にプログラム作りができて OSがいらないの
短所
大きなプログラムが組めない
スピードが遅い
操作性が悪い
16ビット向き
バリバリ
EDIT
長所
大きなプログラムを組むのに適す
短所
OSと大きなメモリが要る
一方 一般によく使われるのはエディタを使う方法で コマンドも豊富で操作性も良く 大きなプログラム組むのに適しているわけ
ただOSと大きなメモリが必要だから 16ビット機種じゃないと 実用的とはいえないの


これらを使って作られたアセンブリ言語のプログラムを 一般にソースプログラムと言います
ふつう ソースプログラムとは 言語処理プログラムの言語仕様に基づいて作られたプログラムのことなの
さてアセンブラを使ってアセンブリ言語で記述されたプログラムをマシン語に変換するのだけど
おお いよいよ核心か！
あ！
そのまえに ひとつ
ソースプログラムをマシン語で実行できるように変換したものを ソースプログラムに対して オブジェクト・プログラムというのよ
アセンブリ言語
マシン語 (インストラクション・コード)
ソース・プログラム
変換
オブジェクトプログラム
エディタを使用して作ります



マシン語基礎用語

アセンブリ言語とアセンブラ

マシン語でプログラムを開発するには，これまでに説明したとおりＣＰＵ独自の命令（インストラクション・コード）で記述しなければなりません。たとえ一歩進んだ高級言語であるＣ言語やＢＡＳＩＣコンパイラにしても，プログラムを記述しているとき，つまりソースリストの段階では我々に理解し得る言語ですが，実行形式のオブジェクトに変換されれば，まぎれもないインストラクション・コードである数値の羅列なのです。しかし，だからといって労力を費やしてインストラクション・コードを覚えるなどということは無駄な努力だといえます。そこで高級言語ほどではありませんが，我々に理解しやすいように考えだされた言語である**アセンブリ言語**の登場となるのです。

さてアセンブリ言語は今流行のＣ言語ほど高級な記述はできませんが，そのかわりといってはなんですが，ＣＰＵ自身のすべてを引き出すことができるような緻密なプログラミングが可能です。しかし，よくよく考えてみればアセンブリ言語は，ＣＰＵのインストラクション・コードに一対一で完全対応しているのですから当り前のことで，長所とはあまりいえないかもしれませんね。さてアセンブリ言語そのものは本文中でも説明しましたが実行形式のオブジェクトに変換するには**アセンブラ**をかけてやらなければなりません。アセンブリ言語で記述されたソースプログラムはアセンブラをかけることにより実行可能なオブジェクトに変換することができるのです。

最近はアセンブリ言語によるプログラム効率をすこしでも改善するべくアセンブラの高級化が進み，**マクロアセンブラ**と呼ばれる大変素晴らしいものがパソコンにも低価格で普及しはじめました。しかし残念ながらこれは16ビット以上のパソコンで，しかもＭＳ－ＤＯＳやＣＰ／Ｍ等の高級なＤＯＳ上に限られています。ですから，本格的なマシン語プログラムを開発するなら必然的に16ビットのマシンが必要不可欠になります。御存じのように16ビットのパソコンはほとんどが日本電気のＰＣ98シリーズ，富士通のＦＭＲシリーズに代表されるようにＣＰＵに8086か80286（相当も含む）を使用しています。ところでこれらのマシンで他の機種のマシン語を開発することができるのでしょうか？　実は**クロスアセンブラ**という変わり種のアセンブラを使えば可能なのです。例えばＰＣ98上でＦＭ77／ＡＶのＣＰＵである6809のオブジェクトを作成することができるのです。

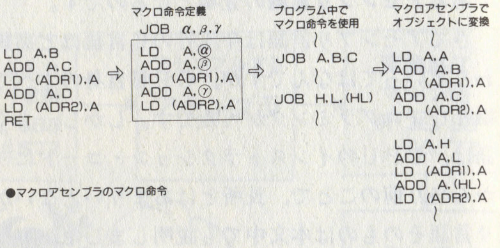
それでは，今までにでてきた専門用語についてまとめておきます。

マクロアセンブラ

一般にアセンブリ言語によるプログラム記述においてマクロ命令が使えるアセンブラのことを総称して呼びます。

マクロ命令

プログラムを記述する過程で同じような処理を行う命令群を一連の文字列で置き換えて，疑似的な命令として扱う処理をさせることができる命令をいいます。このマクロ命令を活用することにより，たとえばパラメータを変えるだけで異なった処理をさせることができ，あたかも汎用性を持たせたようなルーチンを簡単にしかも簡潔に記述することができます。



●マクロアセンブラのマクロ命令

モジュール/モジュラー・プログラミング

大きなプログラムを開発する場合には，一度にすべてのプログラムをせずに，いくつかのブロックに分割し複雑さを分解してプログラムを進めていくのが一般的で妥当です。この分割されたブロックをモジュールと呼び，このようにプログラム作業を進めていく手法をモジュラー・プログラミングと呼びます。

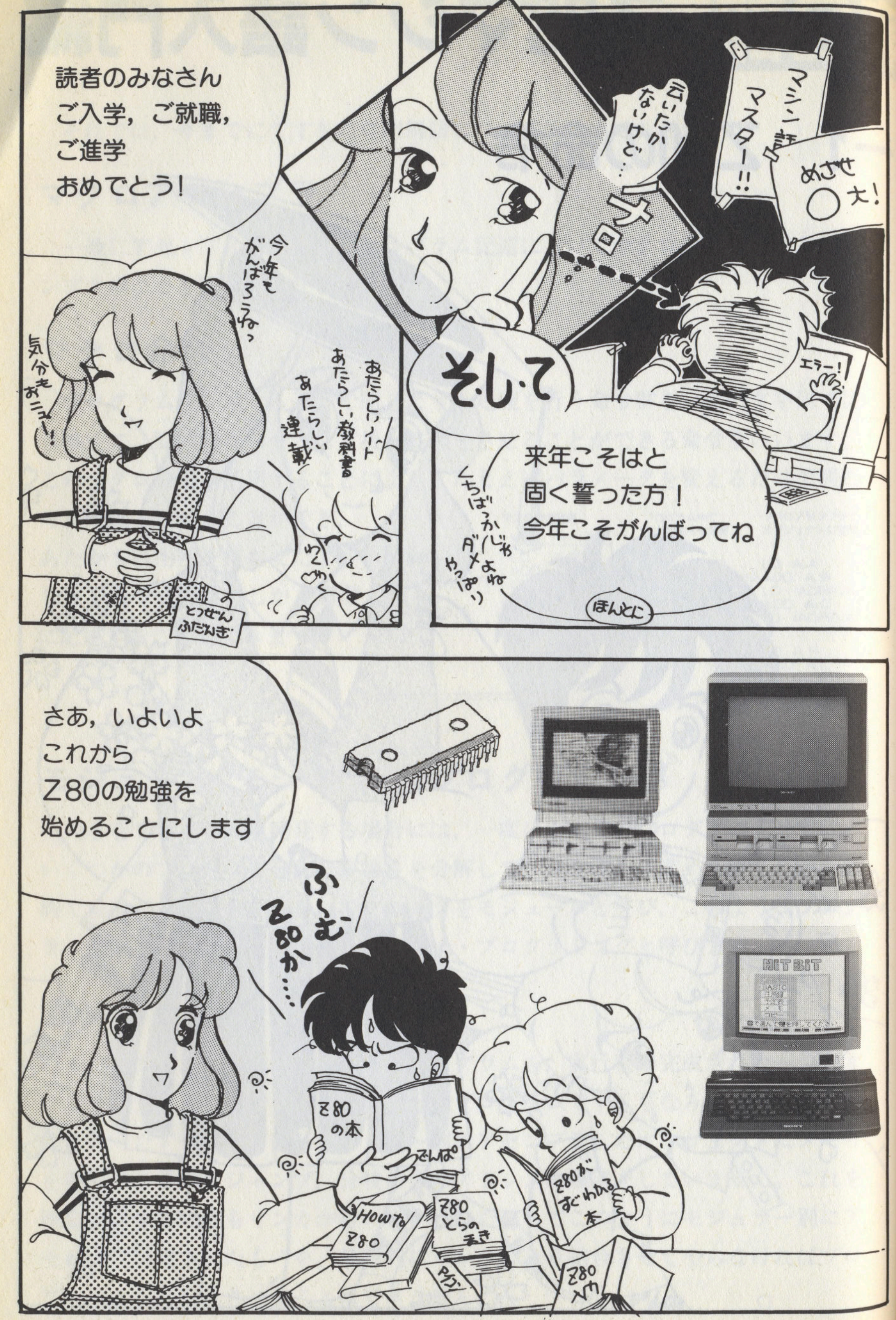
リンカ

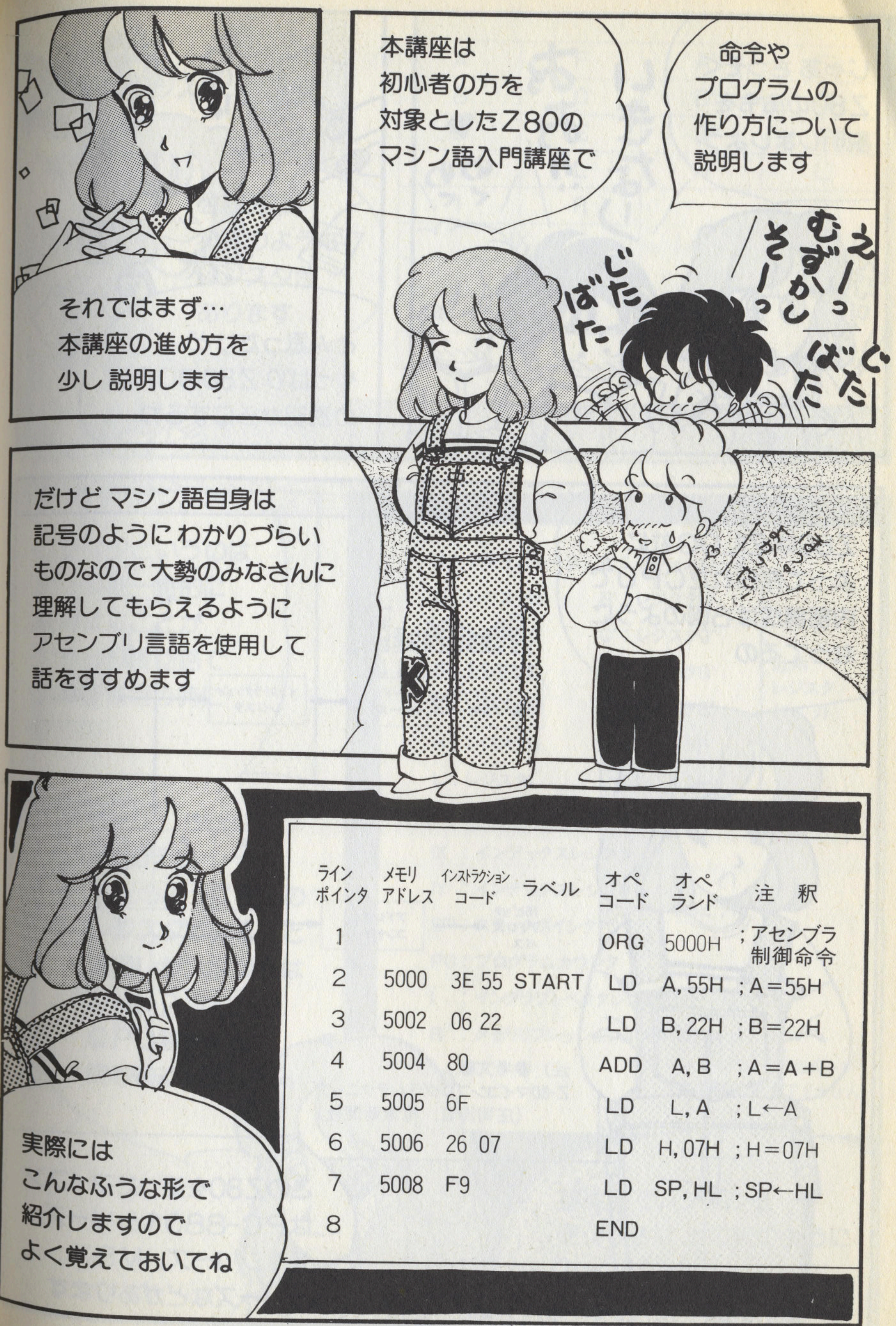
モジュラー別に個別に作成されたプログラムを，実行型の完成された一つのオブジェクトとして結合させる処理を行うプログラムです。ちなみにリロケート機能を持ったアセンブラは，モジュラー別にアセンブラしたオブジェクトにメモリを参照する番地やジャンプ，分岐命令のアドレスの確定をしていません。これを確定してやる処理もリンカが行います。したがってこのようにモジュラー別にアセンブラ処理されたものを，最終的にリンカにより結合させてやらなければプログラムとして実行させることはできません。

第2部　Z80マシン語入門編

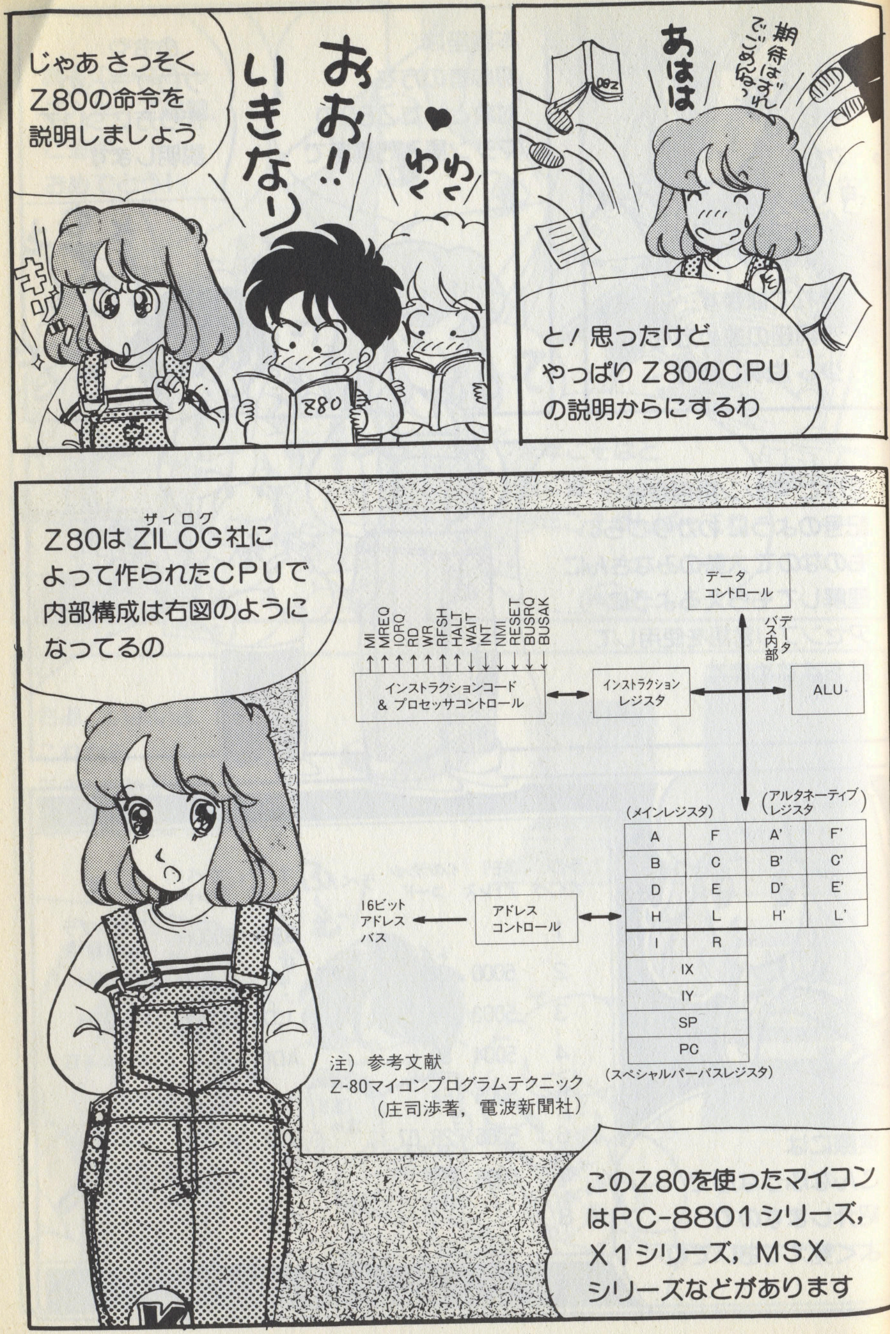
2−1　Z80の命令

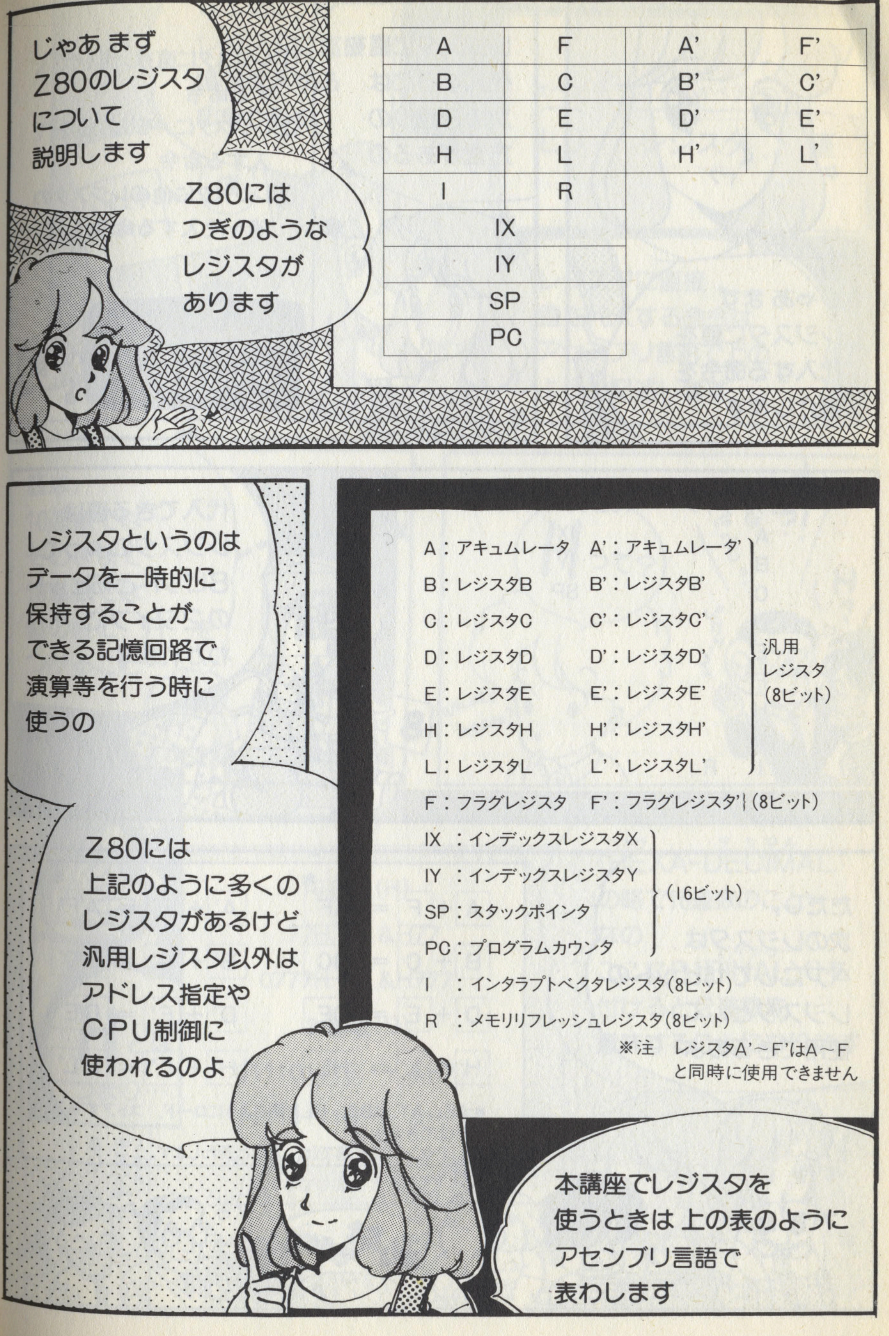


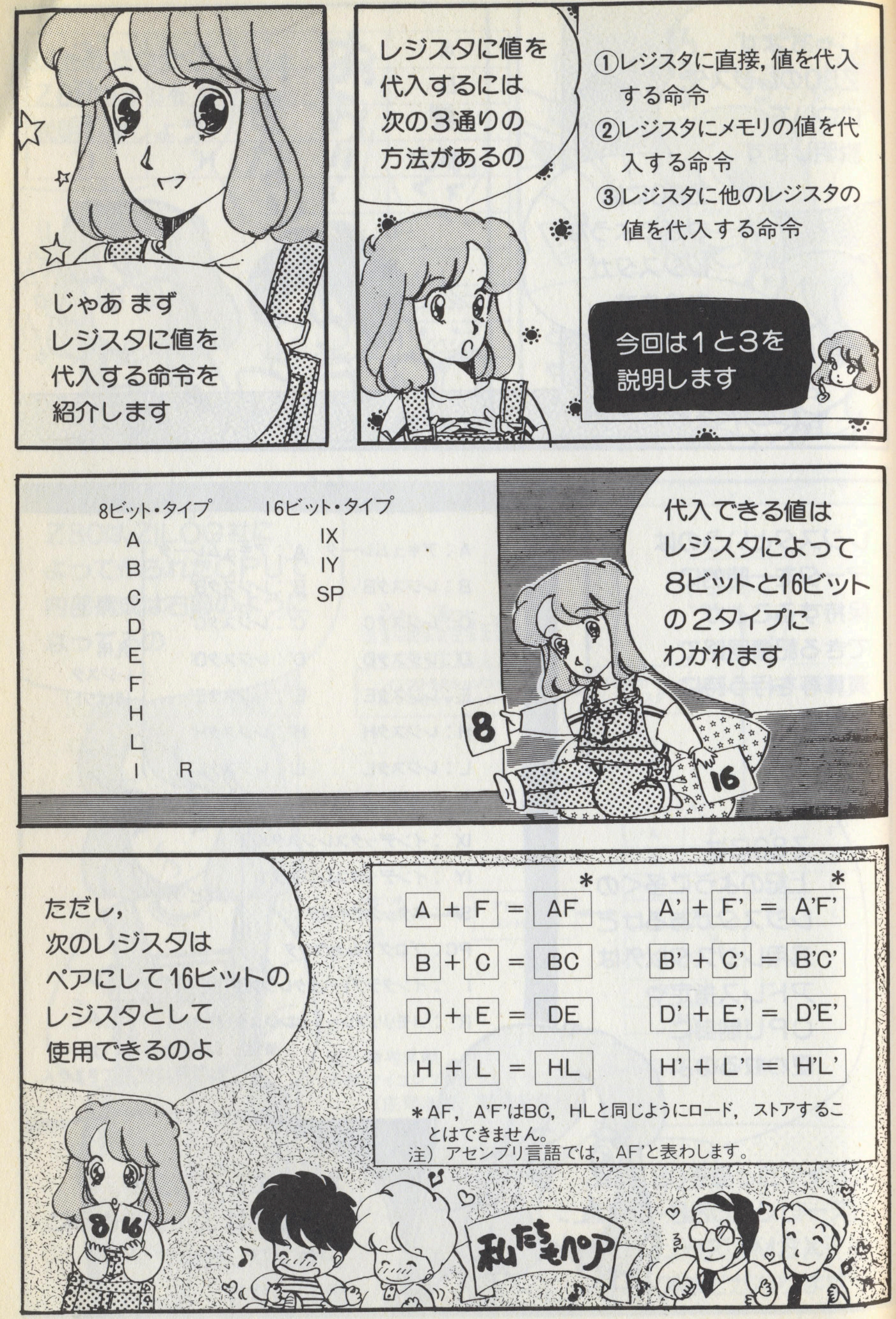




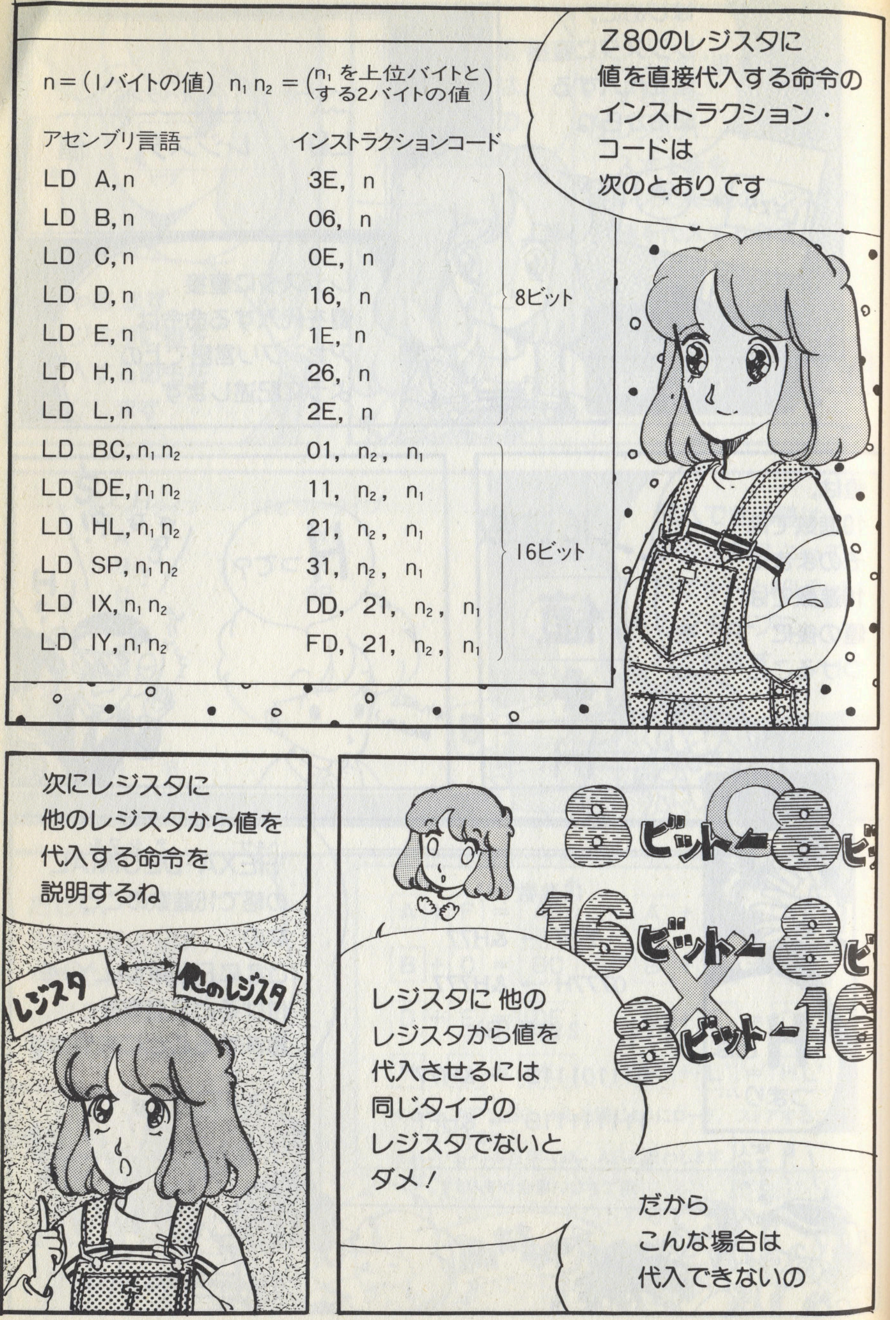
ラインポインタ	メモリアドレス	インストラクションコード	ラベル	オペコード	オペランド	注釈
1				ORG	5000H	;アセンブラ制御命令
2	5000	3E 55	START	LD	A, 55H	;A=55H
3	5002	06 22		LD	B, 22H	;B=22H
4	5004	80		ADD	A, B	;A=A+B
5	5005	6F		LD	L, A	;L←A
6	5006	26 07		LD	H, 07H	;H=07H
7	5008	F9		LD	SP, HL	;SP←HL
8				END		


じゃあ さっそく Z80の命令を 説明しましょう
おお!! いきなり
わくわく
期待はずれでごめんねっ
あはは
と! 思ったけど やっぱり Z80のCPU の説明からにするわ
Z80は ZILOG(ザイログ)社に よって作られたCPUで 内部構成は右図のように なってるの
MI MREQ IORQ RD WR RFSH HALT WAIT INT NMI RESET BUSRQ BUSAK
インストラクションコード & プロセッサコントロール
インストラクション レジスタ
データ コントロール
データ バス内部
ALU
(メインレジスタ)
(アルタネーティブ レジスタ)
A F A' F'
B C B' C'
D E D' E'
H L H' L'
I R
IX
IY
SP
PC
(スペシャルパーパスレジスタ)
16ビット アドレス バス
アドレス コントロール
注) 参考文献
Z-80マイコンプログラムテクニック
(庄司渉著, 電波新聞社)
このZ80を使ったマイコン はPC-8801シリーズ, X1シリーズ, MSX シリーズなどがあります


じゃあ まず Z80のレジスタ について 説明します
Z80には つぎのような レジスタが あります
A F A' F'
B C B' C'
D E D' E'
H L H' L'
I R
IX
IY
SP
PC
レジスタというのは データを一時的に 保持することが できる記憶回路で 演算等を行う時に 使うの
A：アキュムレータ　A'：アキュムレータ'
B：レジスタB　B'：レジスタB'
C：レジスタC　C'：レジスタC'
D：レジスタD　D'：レジスタD'
E：レジスタE　E'：レジスタE'
H：レジスタH　H'：レジスタH'
L：レジスタL　L'：レジスタL'
汎用 レジスタ (8ビット)
F：フラグレジスタ　F'：フラグレジスタ' (8ビット)
IX：インデックスレジスタX
IY：インデックスレジスタY
SP：スタックポインタ
PC：プログラムカウンタ
(16ビット)
I：インタラプトベクタレジスタ(8ビット)
R：メモリリフレッシュレジスタ(8ビット)
※注　レジスタA'～F'はA～F と同時に使用できません
Z80には 上記のように多くの レジスタがあるけど 汎用レジスタ以外は アドレス指定や CPU制御に 使われるのよ
本講座でレジスタを 使うときは 上の表のように アセンブリ言語で 表わします

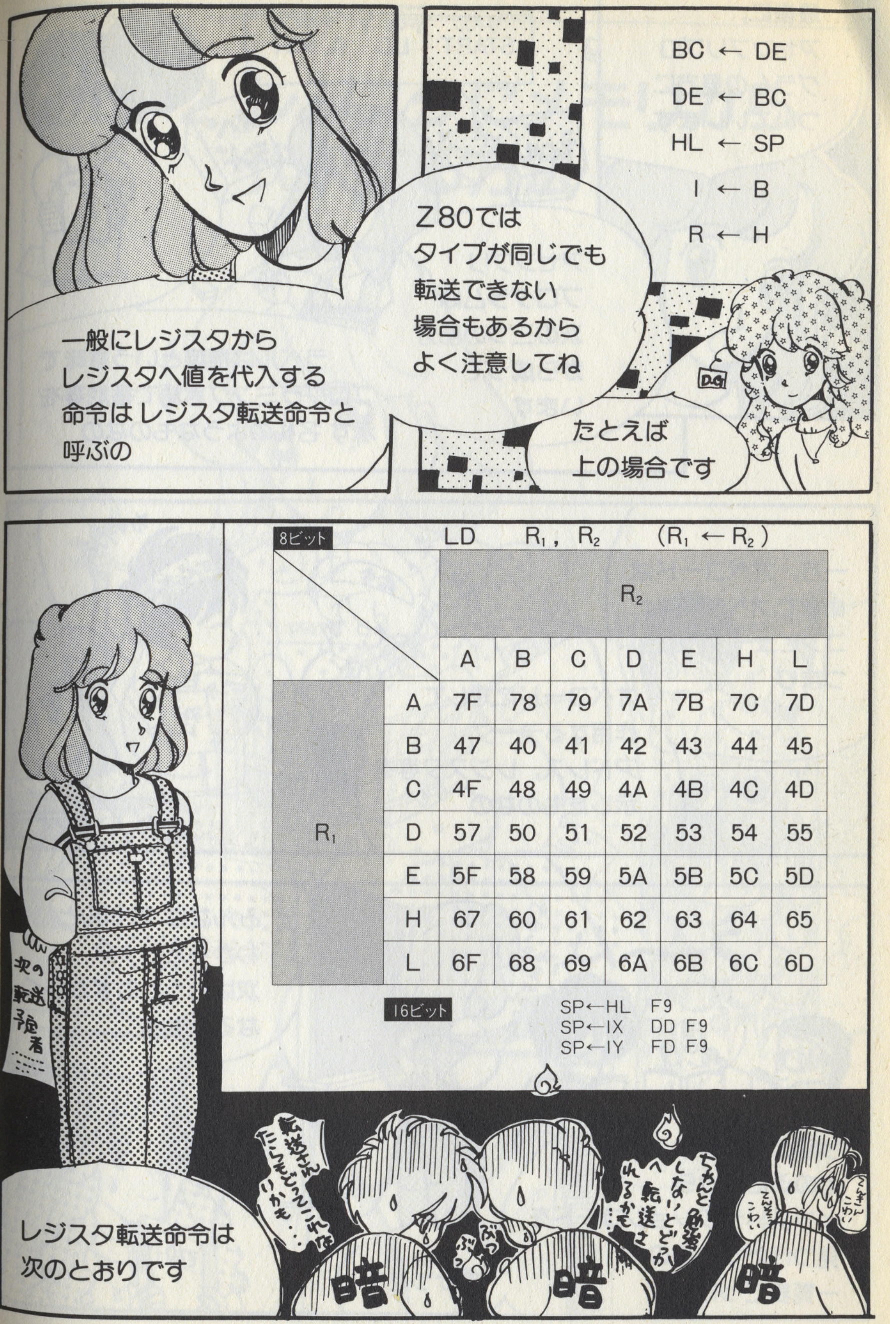

じゃあ まず
レジスタに値を
代入する命令を
紹介します
レジスタに値を
代入するには
次の3通りの
方法があるの
①レジスタに直接，値を代入する命令
②レジスタにメモリの値を代入する命令
③レジスタに他のレジスタの値を代入する命令
今回は1と3を
説明します
8ビット・タイプ　16ビット・タイプ
A　IX
B　IY
C　SP
D
E
F
H
L
I　R
代入できる値は
レジスタによって
8ビットと16ビット
の2タイプに
わかれます
8
16
ただし，
次のレジスタは
ペアにして16ビットの
レジスタとして
使用できるのよ
A + F = AF *　A' + F' = A'F' *
B + C = BC　B' + C' = B'C'
D + E = DE　D' + E' = D'E'
H + L = HL　H' + L' = H'L'
＊AF，A'F'はBC，HLと同じようにロード，ストアすることはできません。
注）アセンブリ言語では，AF'と表わします。
8　16
私たちもペア


はじめに，
レジスタに直接
値を代入する
命令からね
レジスタ ← 値
オペコート　オペランド
LD　レジスタ，値
レジスタに直接
値を代入する命令は
アセンブリ言語で上の
ように記述します
値は，
10進数では
そのままで
16進数では
値の後に“H”を
つけること
16進数
値
＋ プラス
H
Hって？
What's mean?
え?!
Hなはなし？
H？
Hとは
つまり…
人間のぼんのうは百八つ…
16進数(H)
77H → &H77
0777H → &H777
2 進数(B)
01110111B → &H77
11111111B → &HFF
※アセンブラによっては“B”は使用できない場合があります。
“HEXA-DECIMAL”（ヘキサ・デシマル）
の略で16進数のこと
なの
ここにB(BINALY)（バイナリー）を
付けると2進数を
意味することになります
新学期だからリキが入っちゃって…
ほほ…



n =（1バイトの値）　$n_1 n_2$ =（n_1 を上位バイトとする2バイトの値）

アセンブリ言語	インストラクションコード	
LD A, n	3E, n	8ビット
LD B, n	06, n	
LD C, n	0E, n	
LD D, n	16, n	
LD E, n	1E, n	
LD H, n	26, n	
LD L, n	2E, n	
LD BC, $n_1 n_2$	01, n_2, n_1	16ビット
LD DE, $n_1 n_2$	11, n_2, n_1	
LD HL, $n_1 n_2$	21, n_2, n_1	
LD SP, $n_1 n_2$	31, n_2, n_1	
LD IX, $n_1 n_2$	DD, 21, n_2, n_1	
LD IY, $n_1 n_2$	FD, 21, n_2, n_1	



8ビット　LD　R_1, R_2　(R_1 ← R_2)

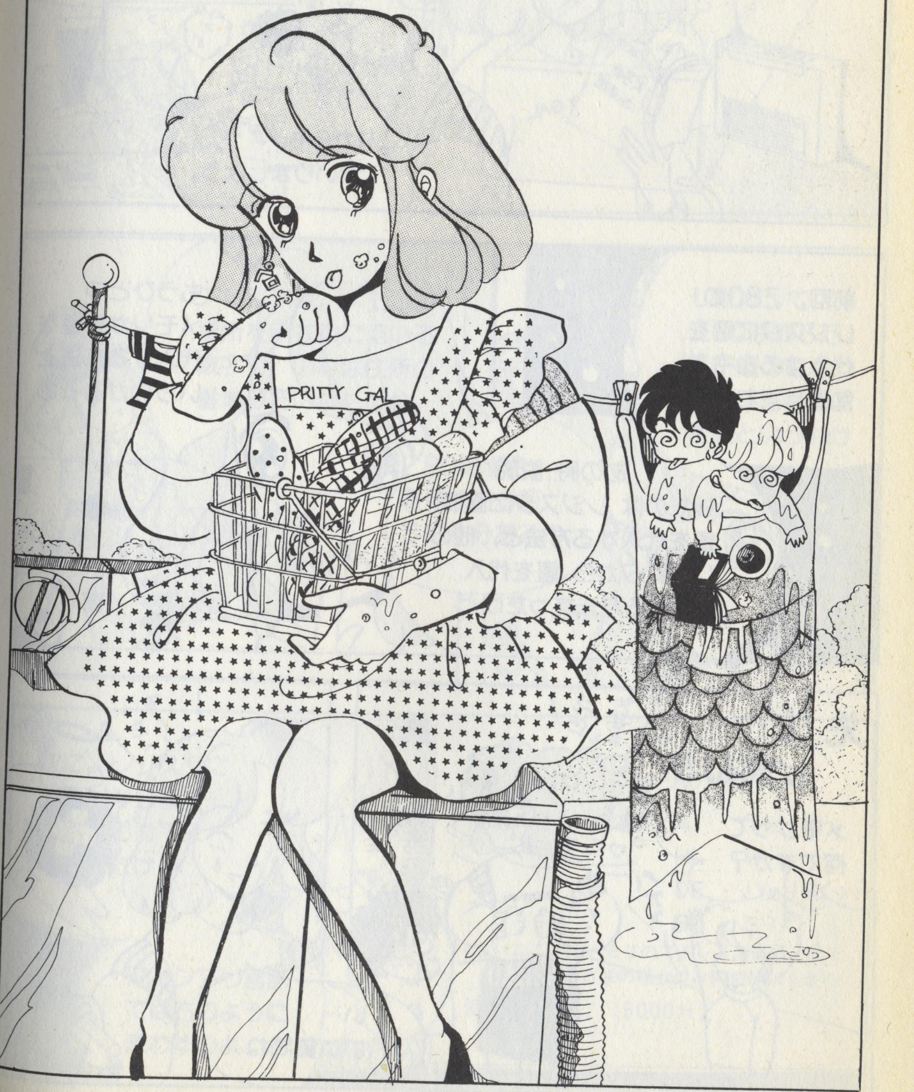
R_1 \ R_2	A	B	C	D	E	H	L
A	7F	78	79	7A	7B	7C	7D
B	47	40	41	42	43	44	45
C	4F	48	49	4A	4B	4C	4D
D	57	50	51	52	53	54	55
E	5F	58	59	5A	5B	5C	5D
H	67	60	61	62	63	64	65
L	6F	68	69	6A	6B	6C	6D

16ビット

SP←HL	F9
SP←IX	DD F9
SP←IY	FD F9

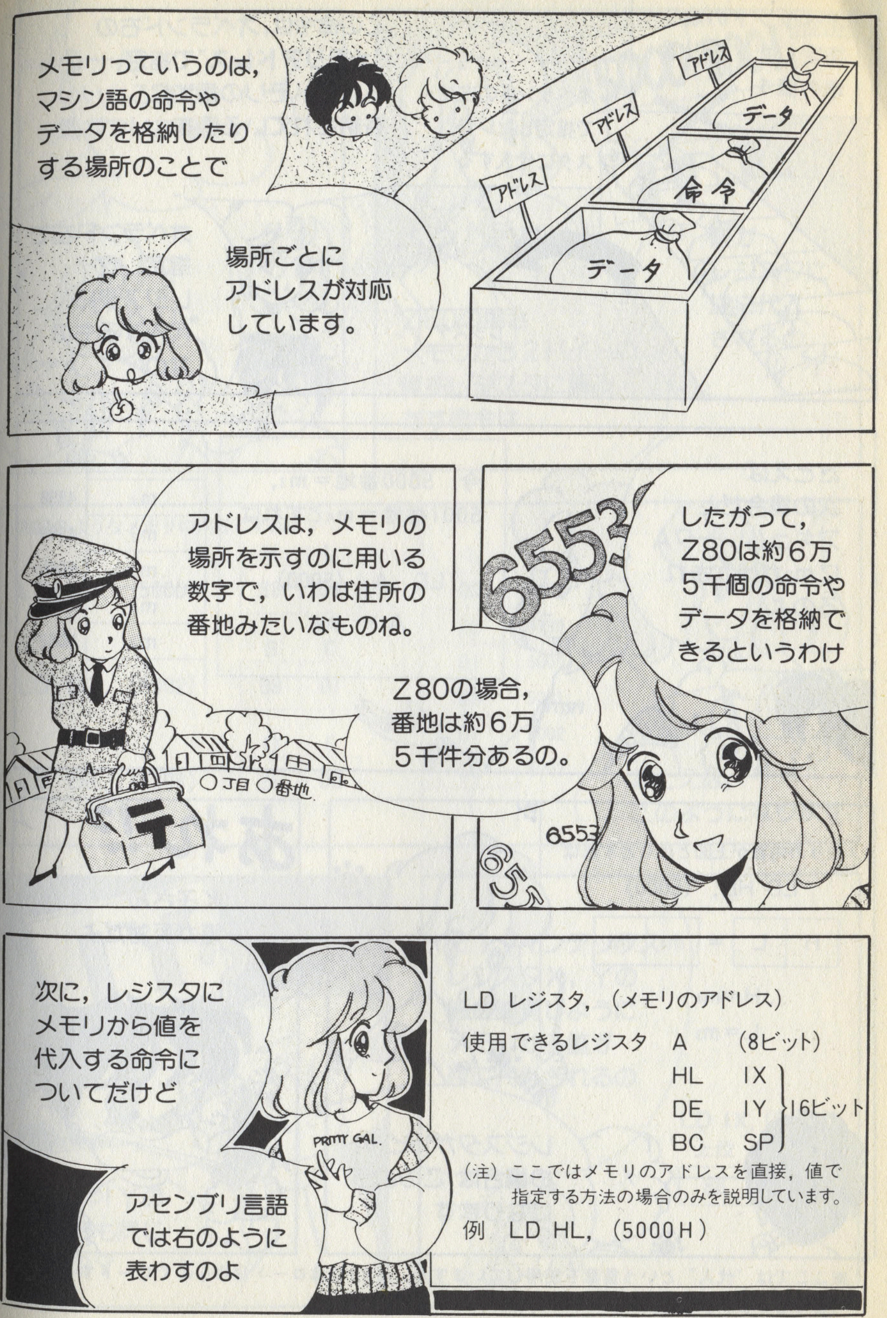


2-2 レジスタにメモリから代入する方法

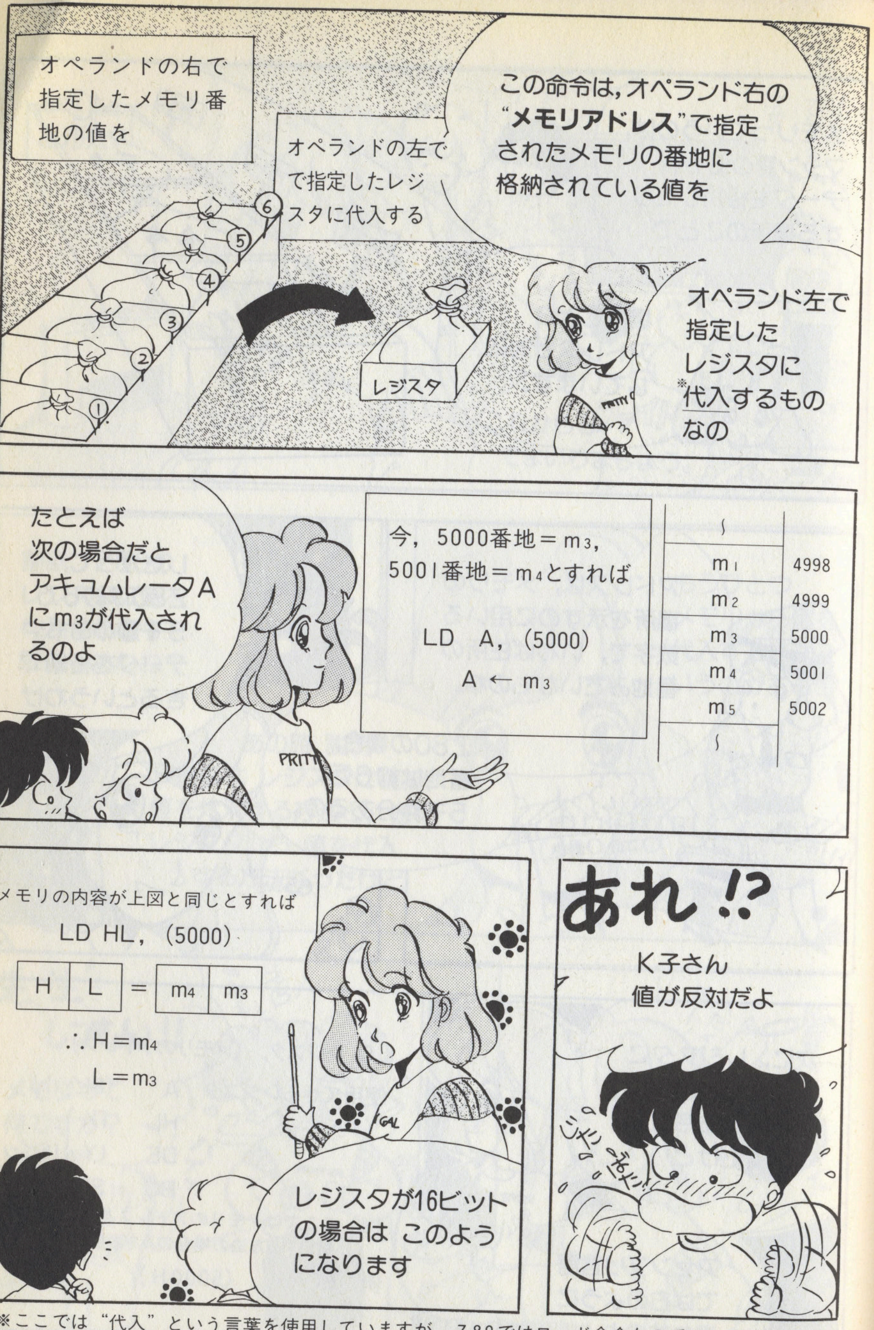




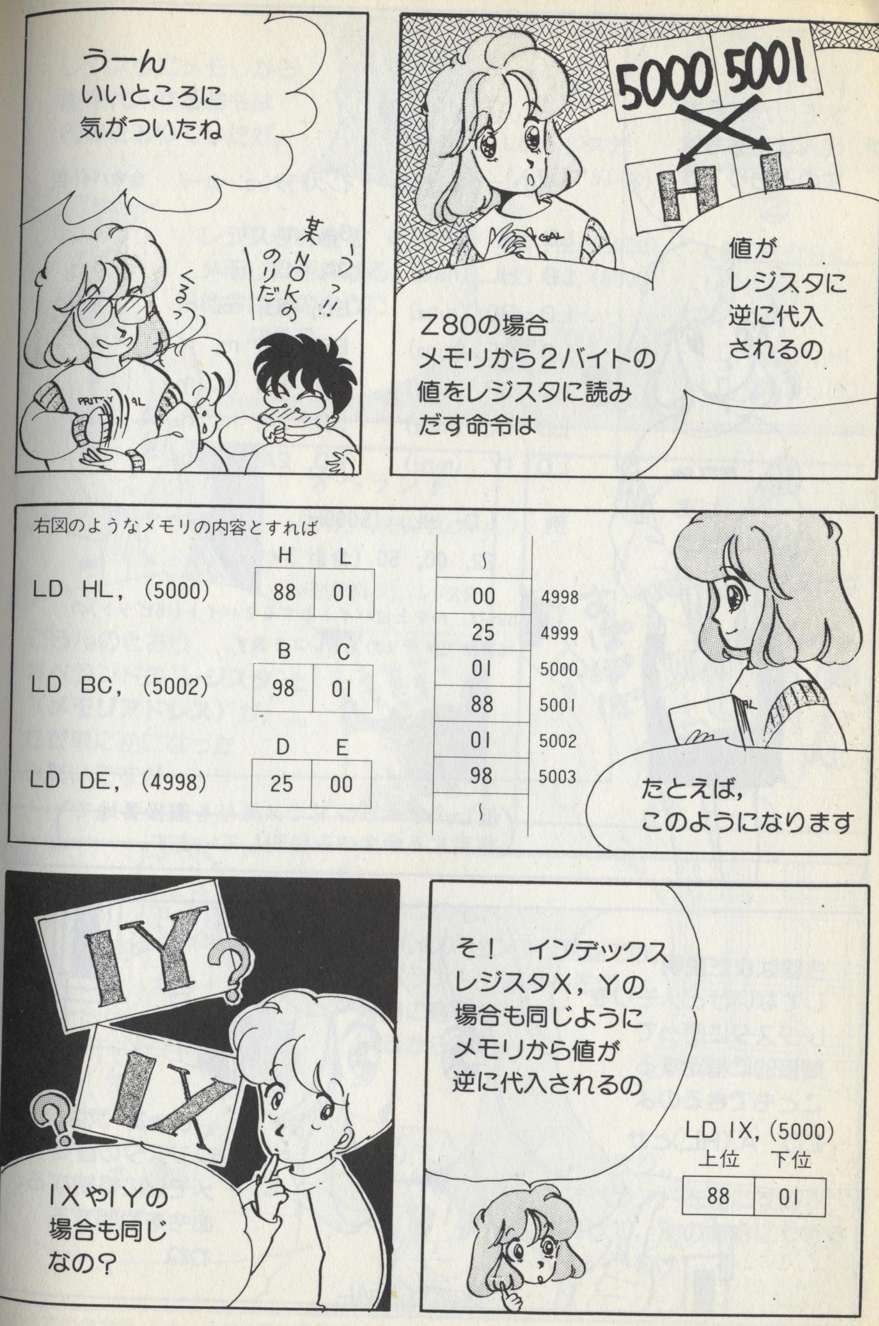
みなさぁーん
お元気？
でしょ？
だから…
わかったよな
わからんよな
これからもファイトでまいりましょう。
前回，Z80のレジスタに値を代入する命令を勉強したわね？
あの時 説明したのは レジスタに直接，値を代入する方法と，他のレジスタから 値を代入する方法だったけど
もうひとつ メモリから値を代入する方法というのがあるの
先生 しつもん!!
メモリって何ですか？
ハイッ
い…いい質問ね

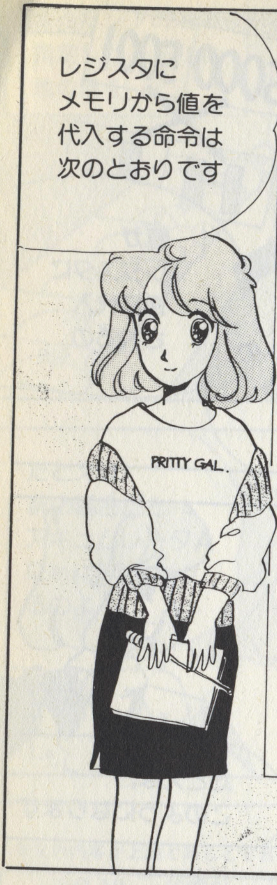


メモリっていうのは，マシン語の命令やデータを格納したりする場所のことで
アドレス
アドレス
アドレス
データ
命令
データ
場所ごとにアドレスが対応しています。
アドレスは，メモリの場所を示すのに用いる数字で，いわば住所の番地みたいなものね。
Z80の場合，番地は約6万5千件分あるの。
○丁目○番地
したがって，Z80は約6万5千個の命令やデータを格納できるというわけ
次に，レジスタにメモリから値を代入する命令についてだけど
アセンブリ言語では右のように表わすのよ
LD レジスタ，（メモリのアドレス）
使用できるレジスタ A （8ビット）
HL IX
DE IY 16ビット
BC SP
（注）ここではメモリのアドレスを直接，値で指定する方法の場合のみを説明しています。
例 LD HL，（5000H）



※ここでは“代入”という言葉を使用していますが，Z80ではロード命令なのでロードするとしてもかまいません。





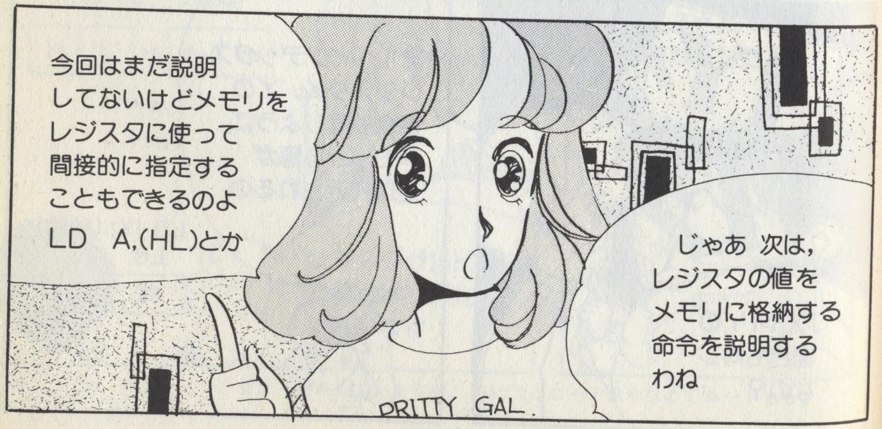
アセンブリ言語	インストラクション・コード	命令バイト数
LD A, (n1n2)(注)	3A, n2, n1	3
LD HL, (n1n2)	2A, n2, n1	3
LD BC, (n1n2)	ED, 4B, n2, n1	4
LD DE, (n1n2)	ED, 5B, n2, n1	4
LD SP, (n1n2)	ED, 7B, n2, n1	4
LD IX, (n1n2)	DD, 2A, n2, n1	4
LD IY, (n1n2)	FD, 2A, n2, n1	4

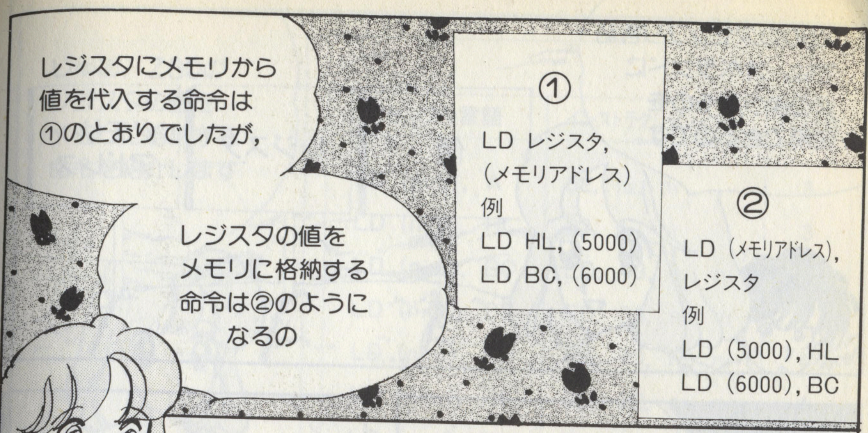
例　LD HL, (5000H)

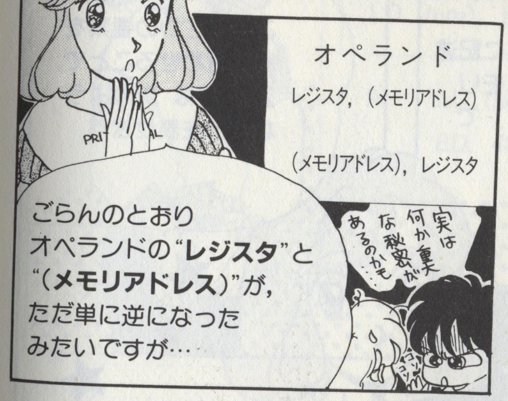
22, 00, 50（合計3バイト）

（注）n1n2は，n1を上位バイトとする2バイト(16ビット)の16進数でメモリのアドレスを表す。

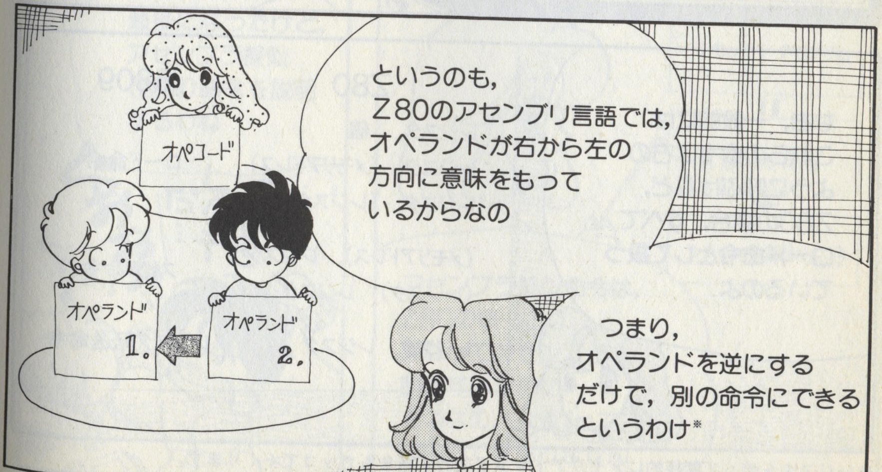
（但し，オペランドでメモリを直接番地で指定する命令のみ説明しています。）



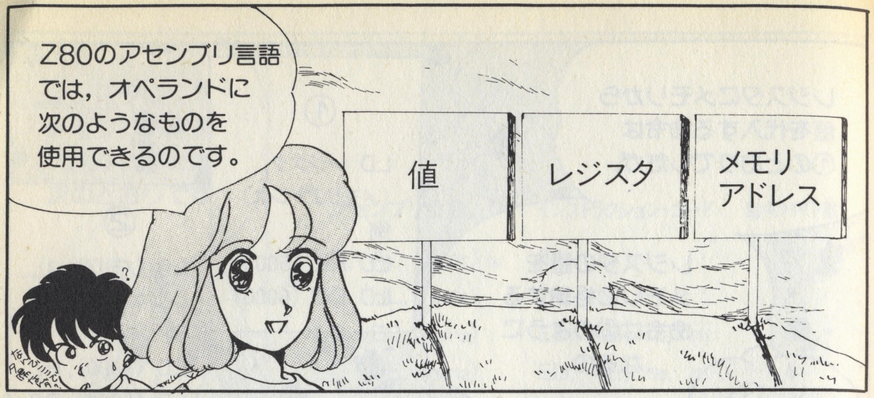


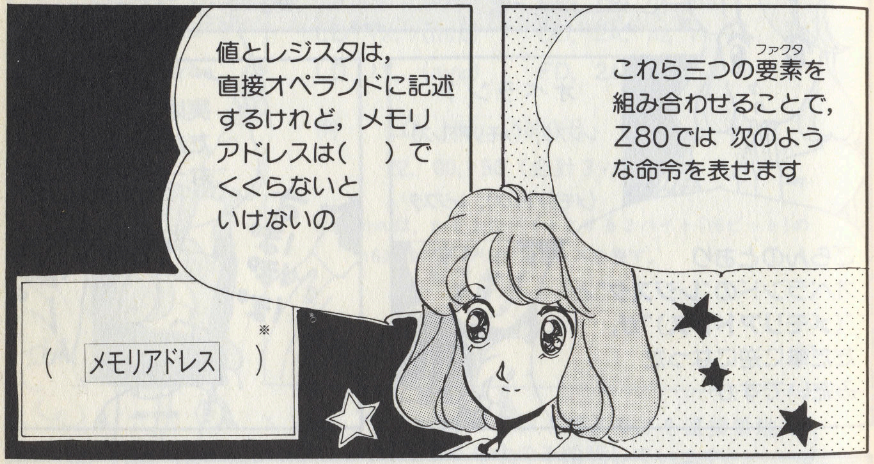


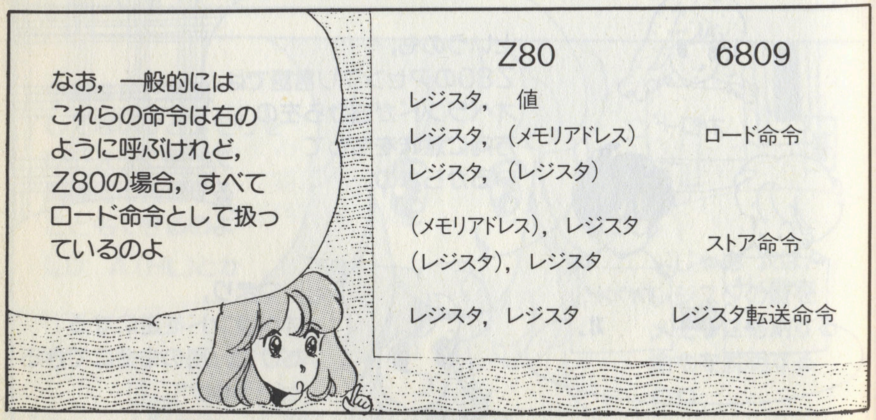




※すべての命令が，そういうわけではありません。

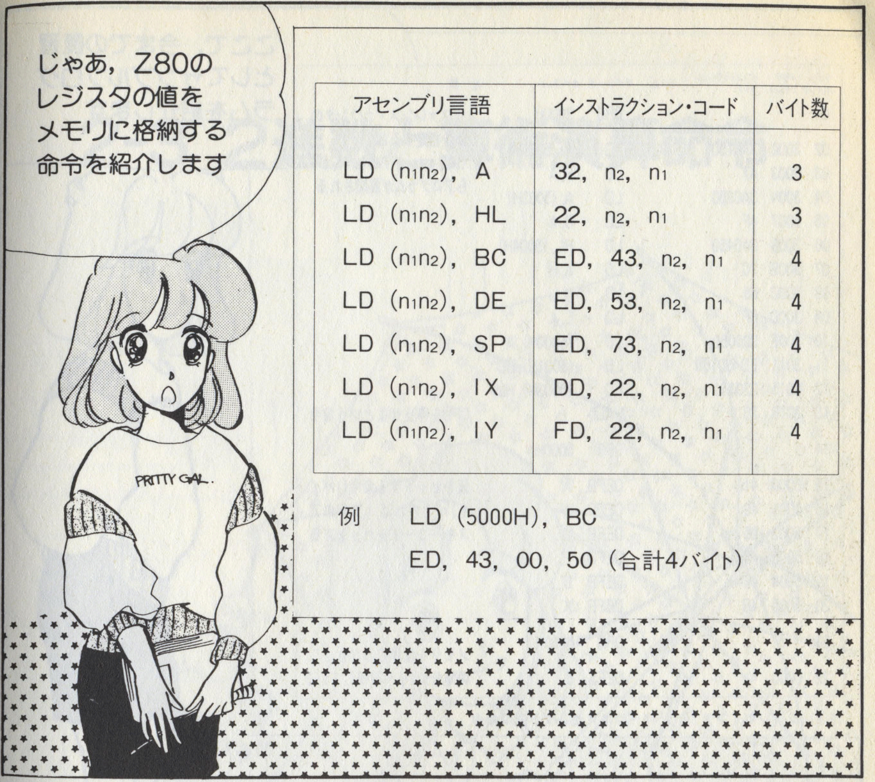






Z80	6809
レジスタ，値 レジスタ，(メモリアドレス) レジスタ，(レジスタ)	ロード命令
(メモリアドレス)，レジスタ (レジスタ)，レジスタ	ストア命令
レジスタ，レジスタ	レジスタ転送命令

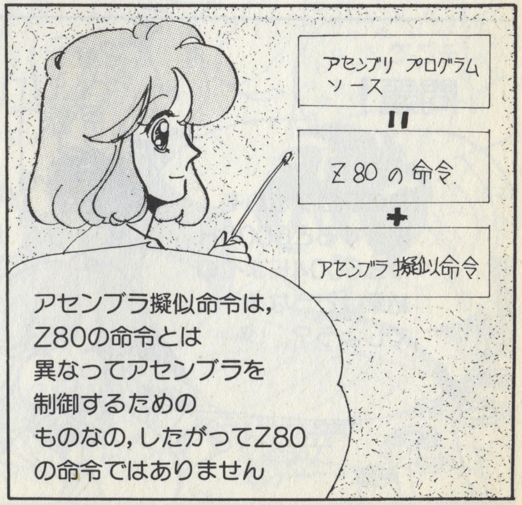
※レジスタを使って間接的にメモリアドレスを表わす場合もカッコでくくります。
例　LD　A,(IX)



アセンブリ言語	インストラクション・コード	バイト数
LD (n_1n_2), A	32, n_2, n_1	3
LD (n_1n_2), HL	22, n_2, n_1	3
LD (n_1n_2), BC	ED, 43, n_2, n_1	4
LD (n_1n_2), DE	ED, 53, n_2, n_1	4
LD (n_1n_2), SP	ED, 73, n_2, n_1	4
LD (n_1n_2), IX	DD, 22, n_2, n_1	4
LD (n_1n_2), IY	FD, 22, n_2, n_1	4

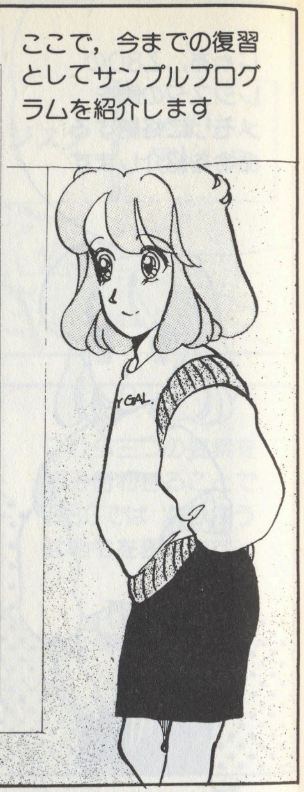
例　LD (5000H), BC
ED, 43, 00, 50 (合計4バイト)

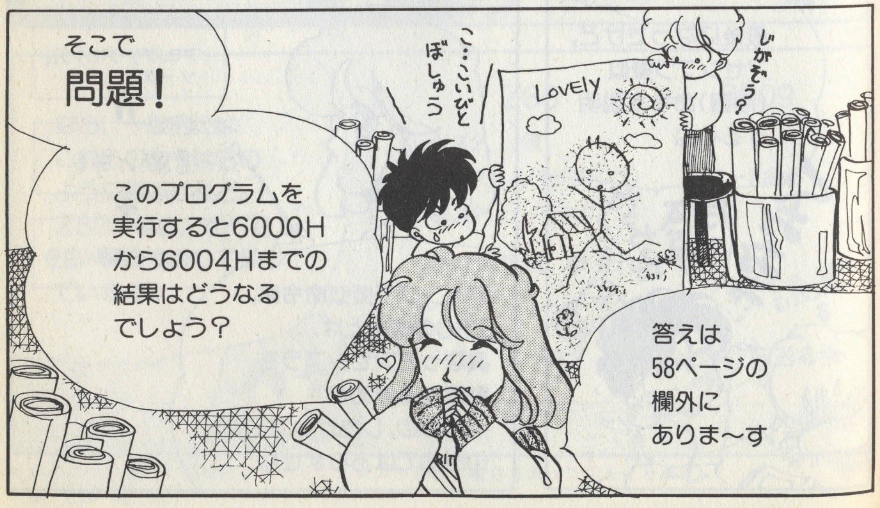




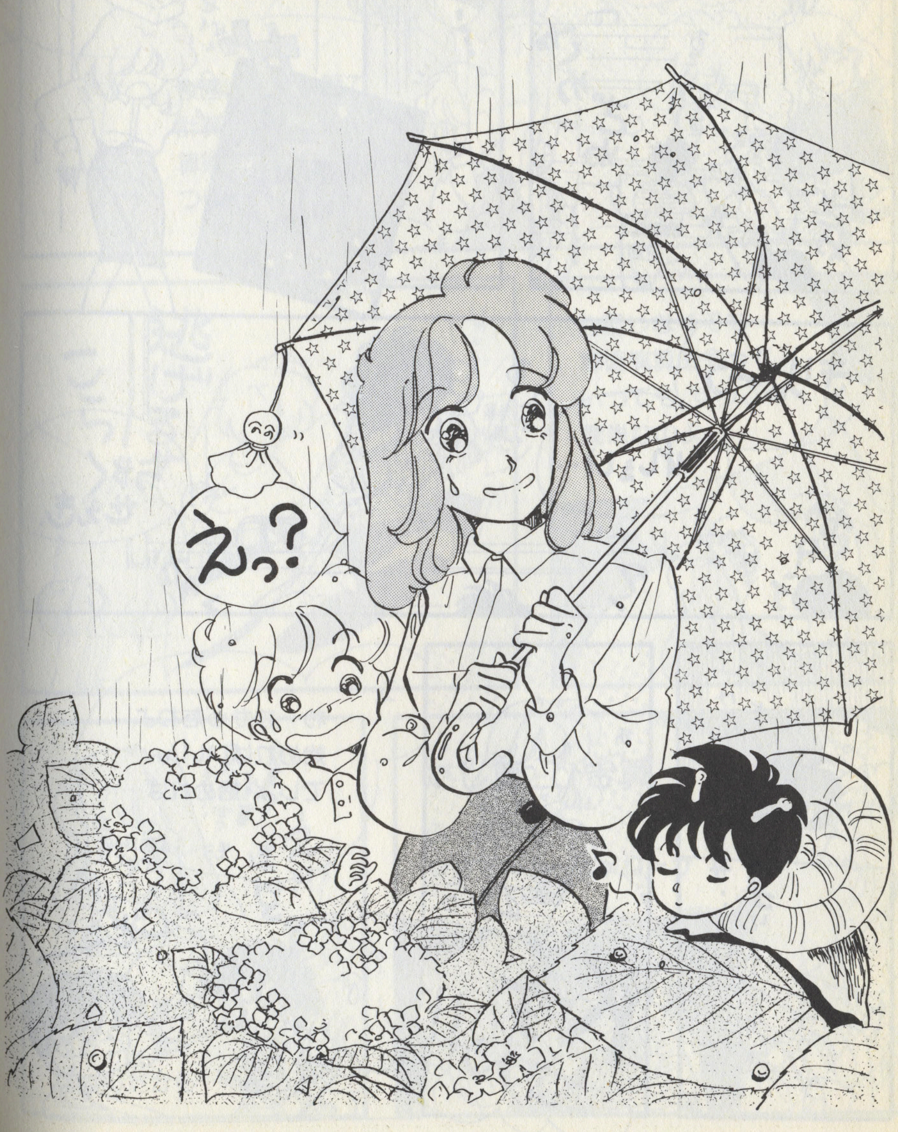
ラインポインタ	メモリアドレス	インストラクションコード	ラベル	オペコード	オペランド	注釈
01				ORG	3000H	；リファレンス・カウンタの値を指定する命令 ORG n_1n_2とするとn_1n_2からプログラムが展開される
02	3000	3A0150	START:	LD	A, (5001H)	
03	3003	47		LD	B, A	
04	3004	3A0350		LD	A, (5003H)	
05	3007	4F		LD	C, A	
06	3008	2A0450		LD	HL, (5004H)	
07	300B	7C		LD	A, H	
08	300C	65		LD	H, L	
09	300D	6F		LD	L, A	
10	300E	320060		LD	(6000H), A	
11	3011	ED430160		LD	(6001H), BC	
12	3015	220360		LD	(6003H), HL	
13	3018	76		HALT		；CPUを停止せよという命令
14				ORG	5000H	
15	5000	41		DEFB	'A'	；値をセットする命令(1バイト)'n'とすれば1文字のアスキーコードをセットする
16	5001	49		DEFB	'I'	
17	5002	55		DEFB	'U'	
18	5003	45		DEFB	'E'	
19	5004	4F		DEFB	'O'	
20	5005	4B		DEFB	'K'	
21				ORG	6000H	メモリを確保する命令。こ
22	6000			DEFS	5	；の場合5バイト確保している

(注) ラインポインタ 01, 14～22はアセンブラ擬似命令





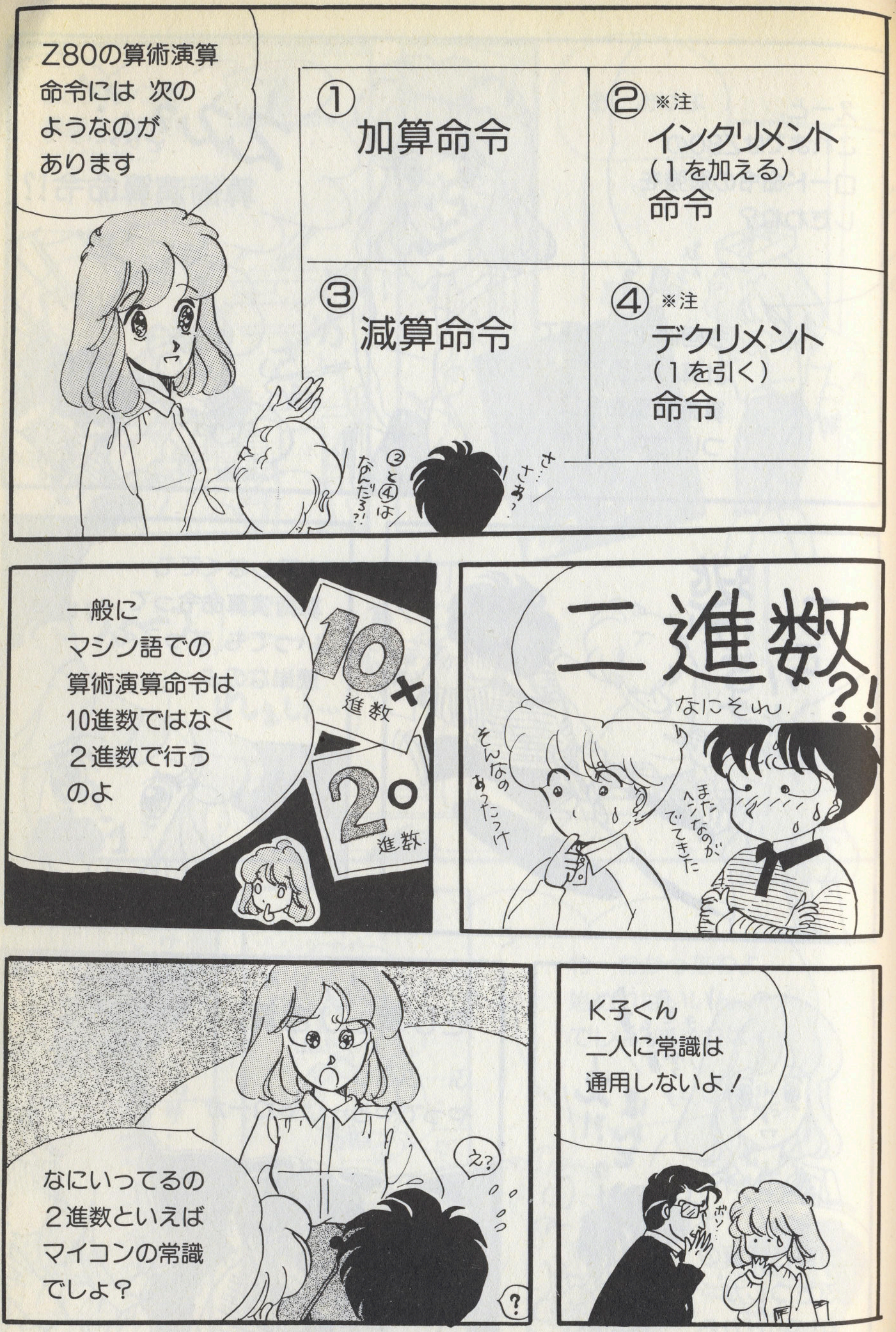
2-3 2進数と算術演算命令



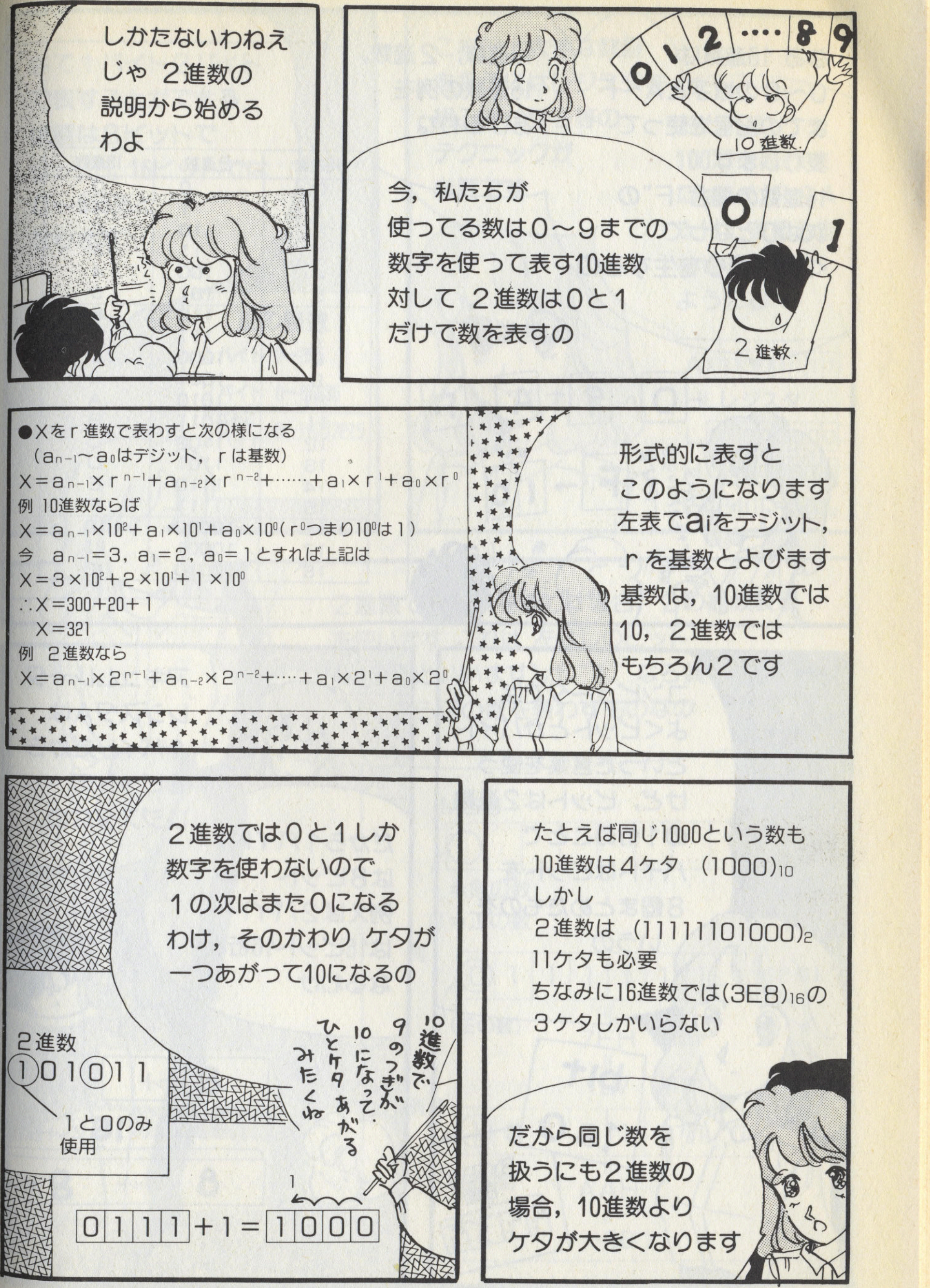


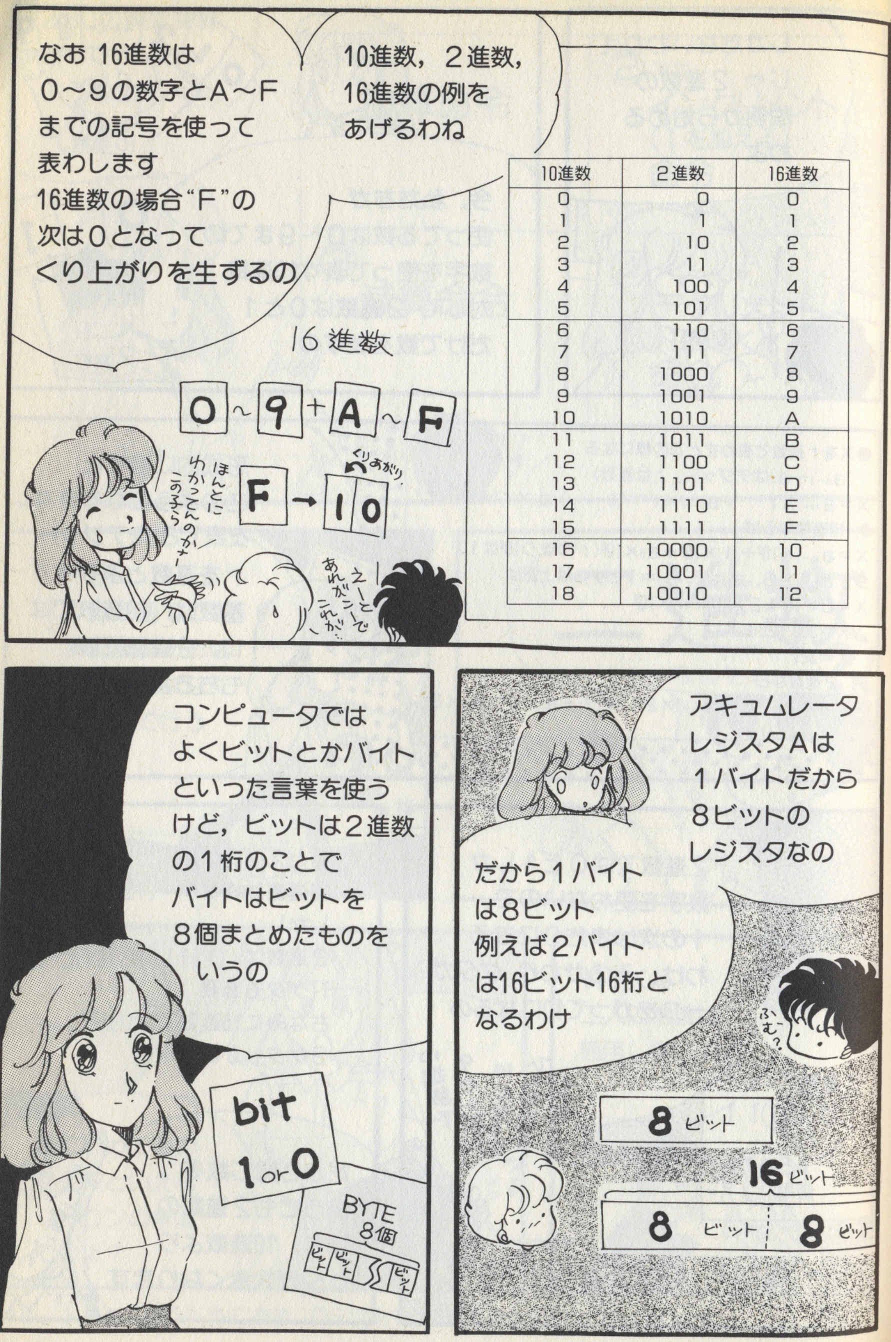


●56ページの答え●6000番地から順にK, E, I, K, O

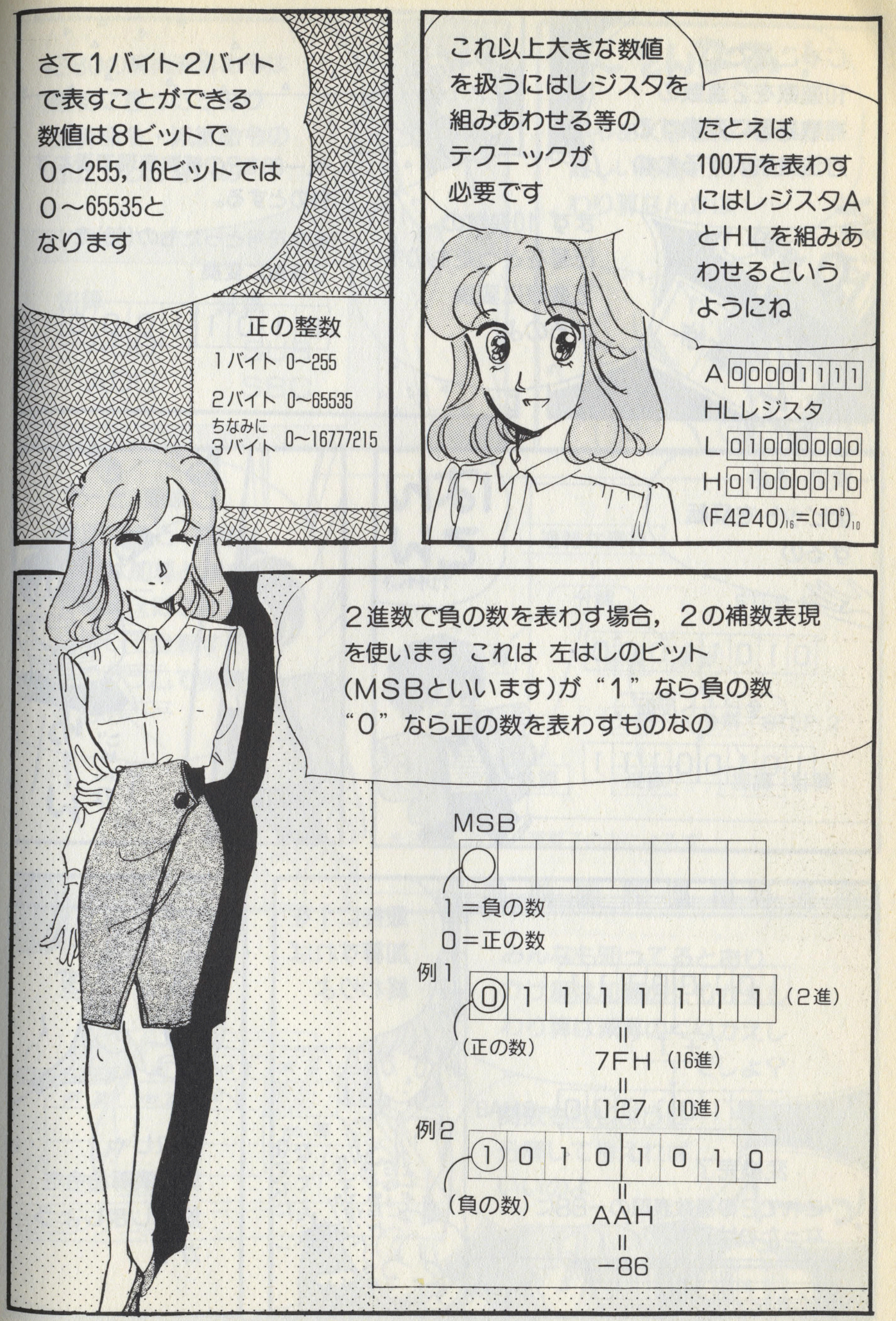


※ここでは INC命令，DEC命令も算術演算命令として扱っています。



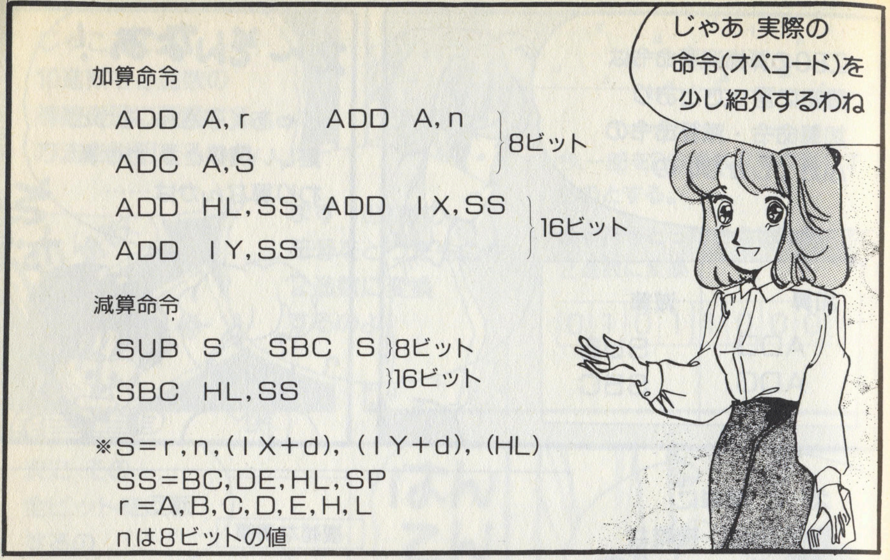


10進数	2進数	16進数
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10
17	10001	11
18	10010	12

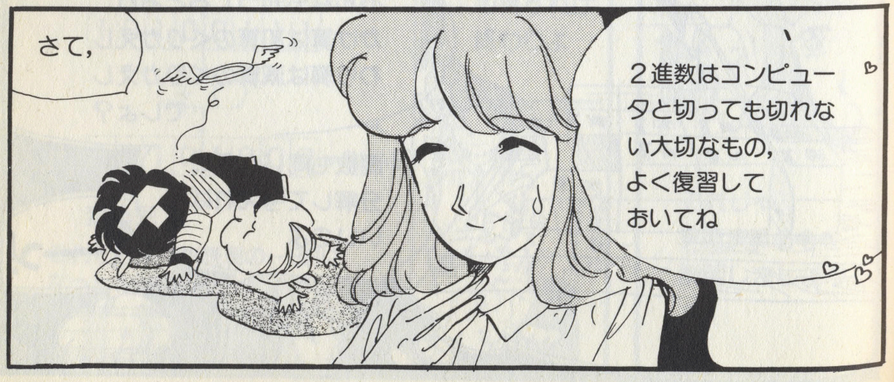



じゃここで
10進数を2進数の
補数表現に変換する
方法を説明するわね
10進数
2の補数
まず 10進数の
符号をとったものを
2進数に変換
するのよ
今,−88を2の補数表現で表わすものとする。
まず符号とったもの(88)を
2進数に変換
01011000
次に その
全ビットを反転
するの
01011000
全ビット反転
10100111
はんてん
ナンチャッテ
ゆーん
ごめんな
さーい
10100111
+1
10101000 =&HA8
変換完了
これで2の補数表現の−88に
なったわけです。
最後に1を
加算すれば
終わりよ
なすびに
なっちった
なすびに反
転したまま
のA君
それじゃ
算術演算命令を
説明しましょう
か！


Z80の算術演算命令は
最初に言ったとおり
加算命令・減算命令の
ふたつしかないの
加算
ADD
ADC
減算
SUB
SBC
え〜そんなあ！
じゃあK子さん もっと
難しい関数とか,かけ算
わり算なんかは……
そだ
えーと 簡単に
説明すると 複雑な
演算は加算と減算の
プロセスにこまかく
分解して計算する
プログラムで実現
するの
複雑な演算
分解
STEP 1
※注
乗算
除算
乗算
分解
分解
分解
STEP 2
加算
減算
減算
加算
計算
※注 は単一演算子のみによる式
みんなも知ってるとおり
かけ算は加算のくりかえし
わり算は減算のくりかえし
でしょ？
関数も同じように
分解して考えれば
いいのよ
へーっ！

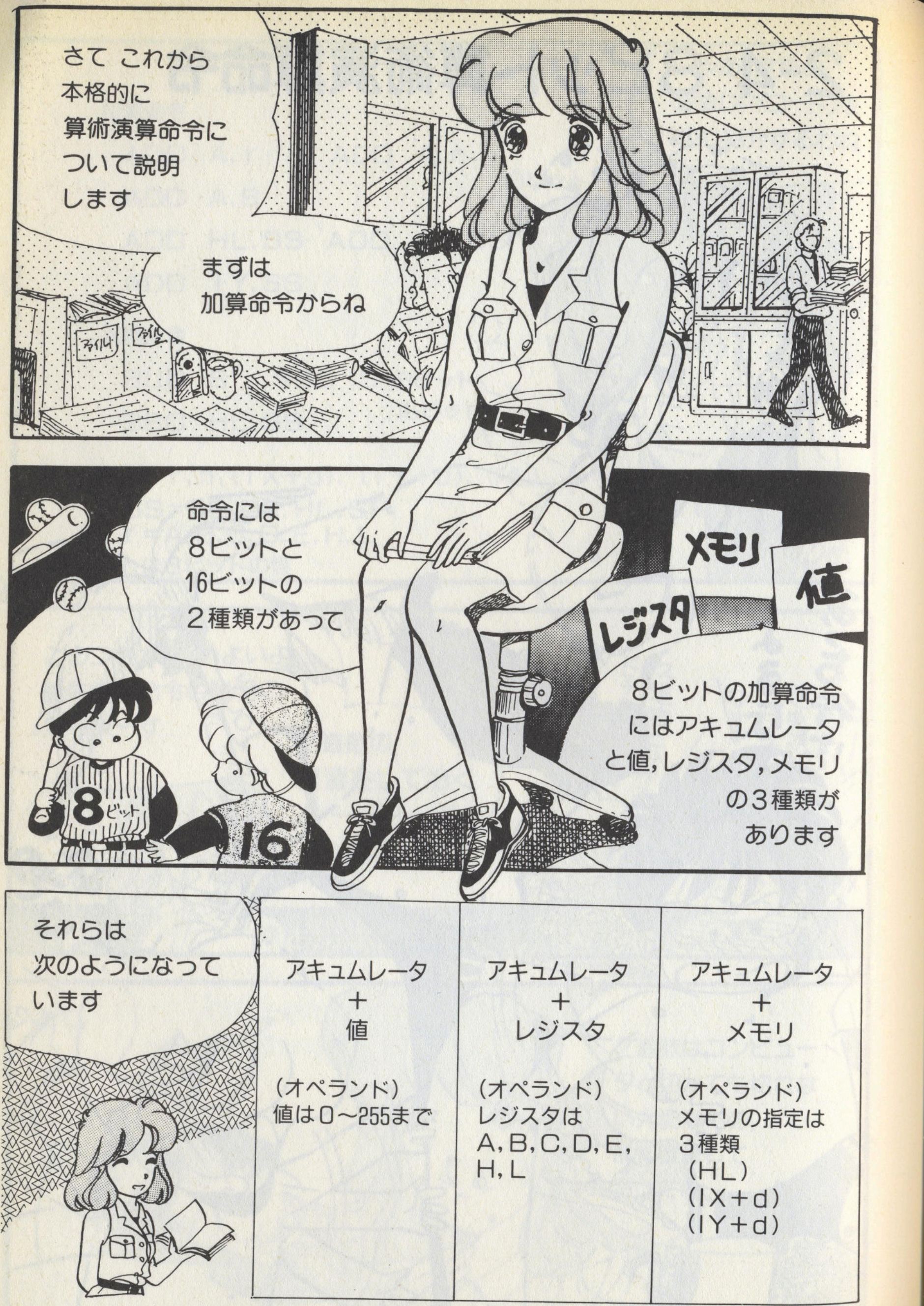




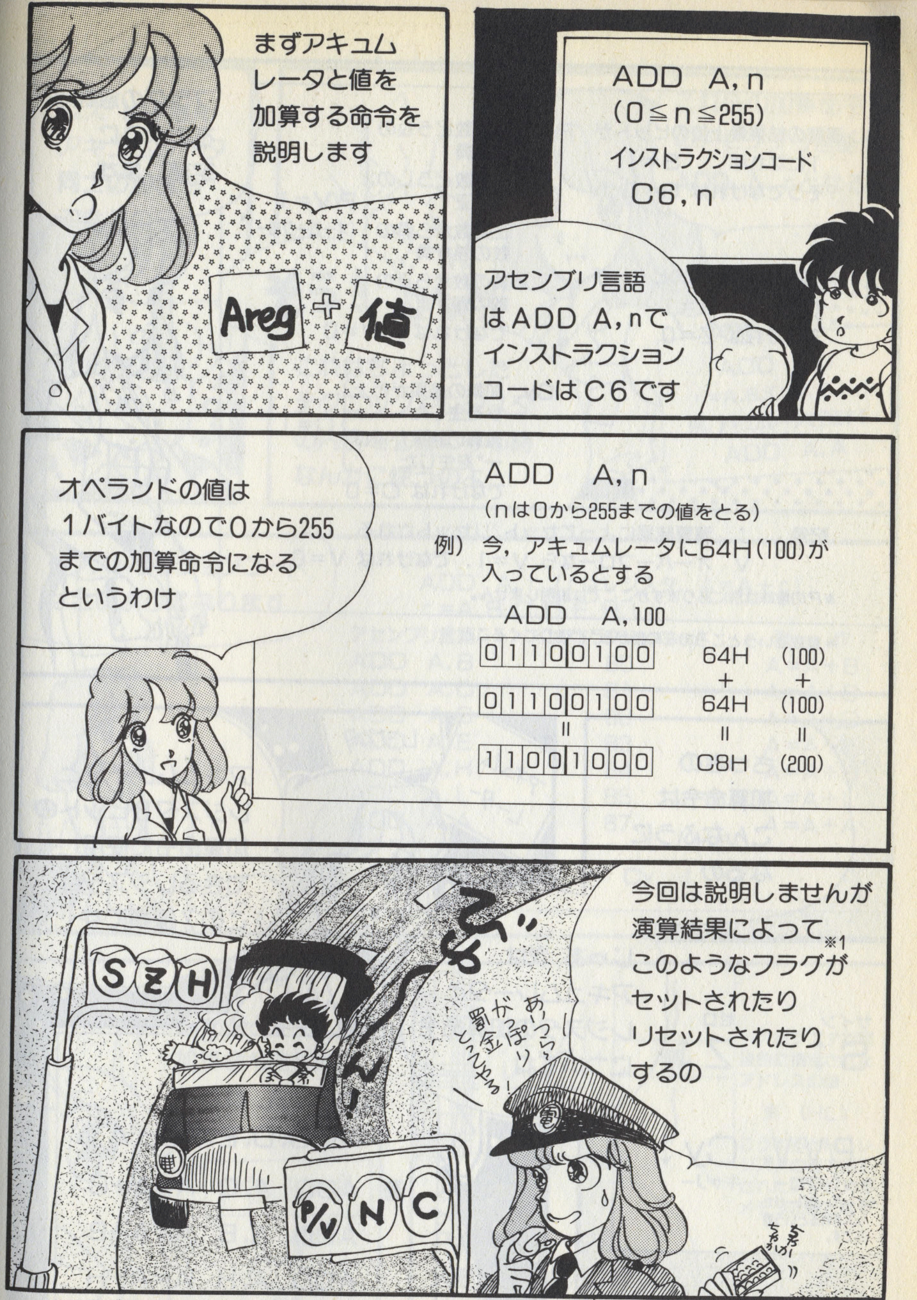


2-4 8ビット算術演算命令



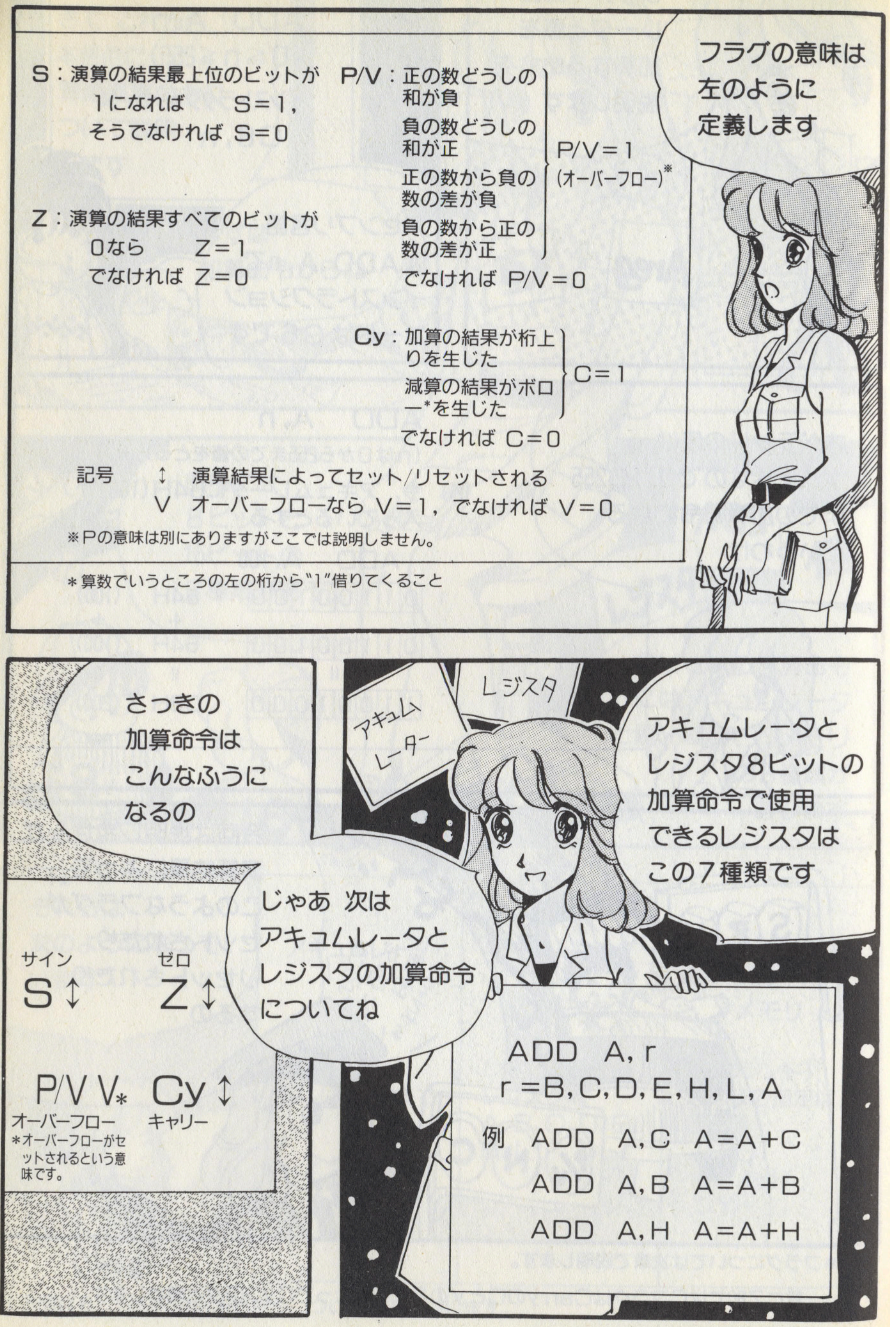


※今回は8ビットの算術演算命令についてのみ説明します。

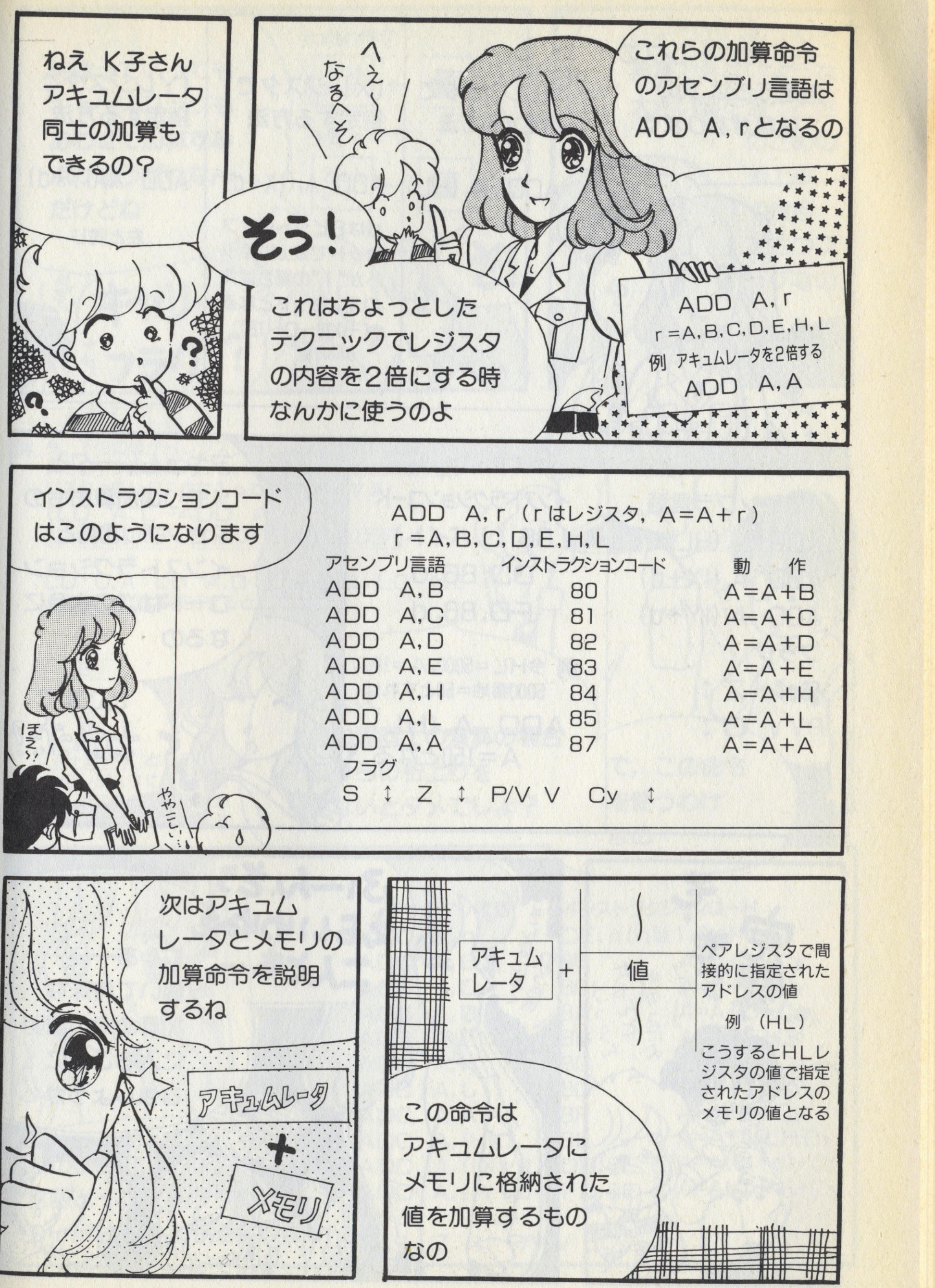


※フラグについては次章で説明します。

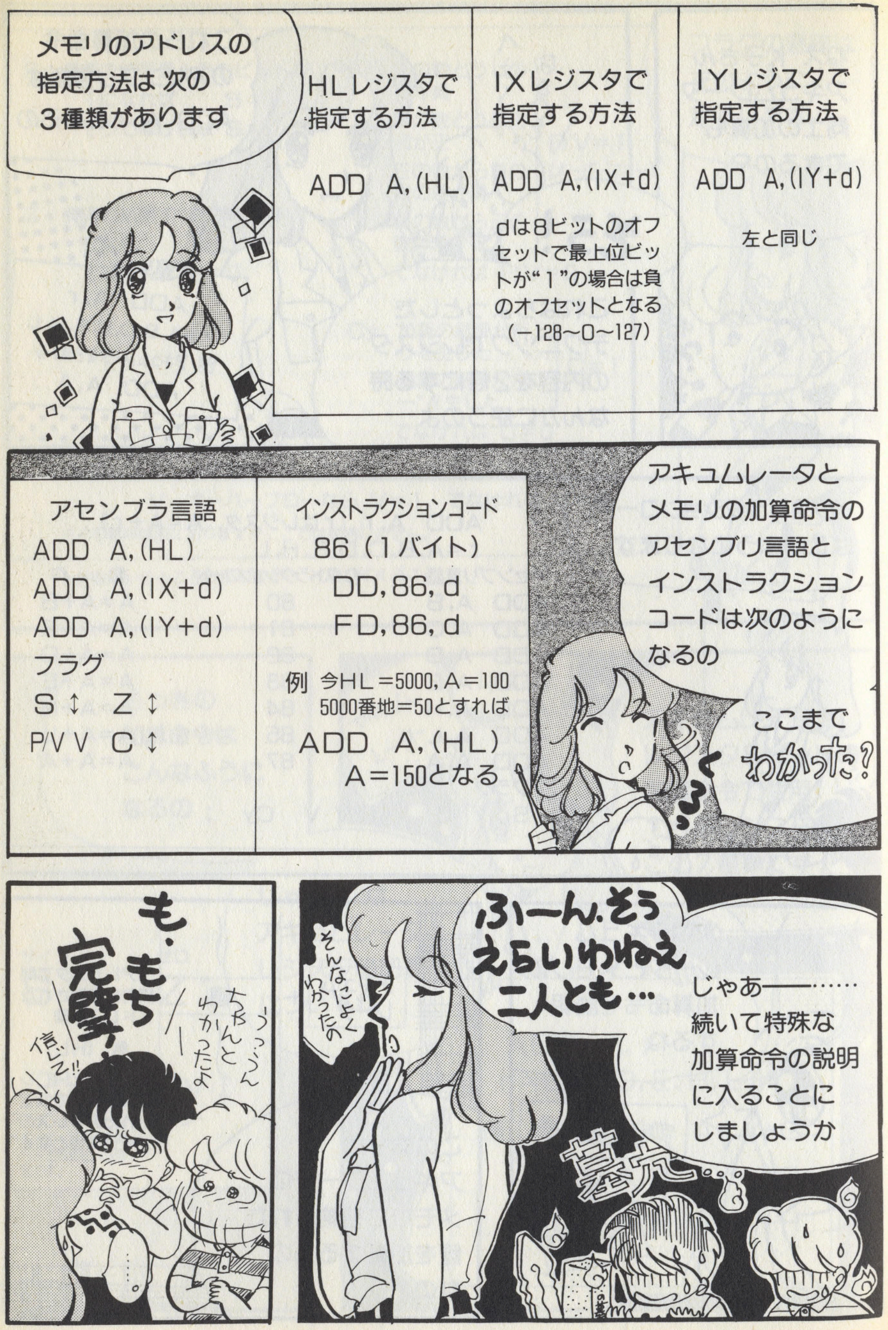
※1 一般にZ80ではCフラグはCarryのCとyの2文字をとってCyフラグと表わします。

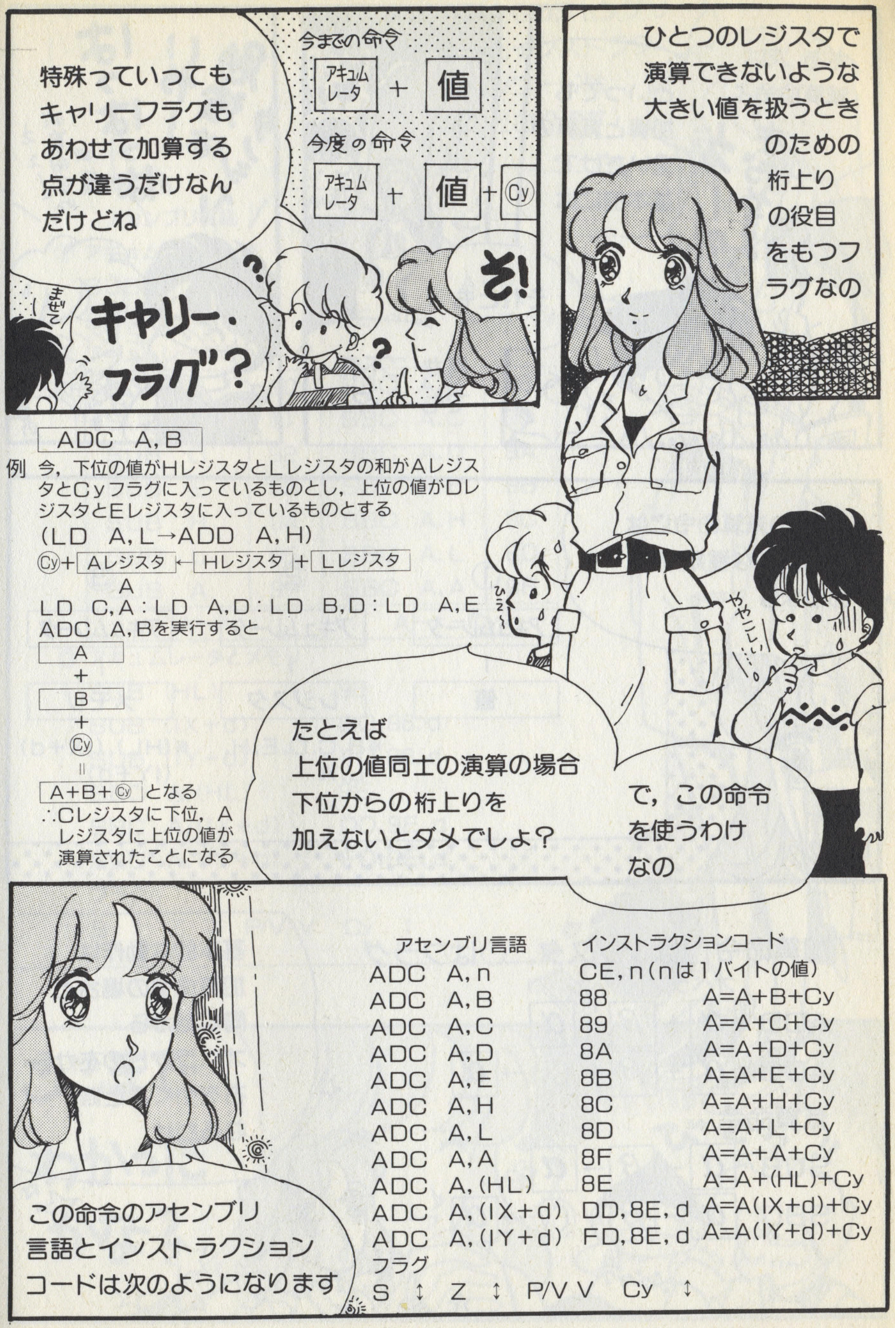


S：演算の結果最上位のビットが
1になれば　S=1,
そうでなければ　S=0
Z：演算の結果すべてのビットが
0なら　Z=1
でなければ　Z=0
P/V：正の数どうしの和が負
負の数どうしの和が正
正の数から負の数の差が負
負の数から正の数の差が正
P/V=1
(オーバーフロー)*
でなければ　P/V=0
Cy：加算の結果が桁上りを生じた
減算の結果がボロー*を生じた
C=1
でなければ　C=0
記号　↕　演算結果によってセット/リセットされる
V　オーバーフローなら V=1，でなければ V=0
＊Pの意味は別にありますがここでは説明しません。
＊算数でいうところの左の桁から"1"借りてくること
フラグの意味は左のように定義します
さっきの加算命令はこんなふうになるの
サイン
S↕
ゼロ
Z↕
P/V V*
オーバーフロー
Cy↕
キャリー
＊オーバーフローがセットされるという意味です。
アキュムレータ
レジスタ
アキュムレータとレジスタ8ビットの加算命令で使用できるレジスタはこの7種類です
じゃあ 次はアキュムレータとレジスタの加算命令についてね
ADD A, r
r=B,C,D,E,H,L,A
例 ADD A,C A=A+C
ADD A,B A=A+B
ADD A,H A=A+H

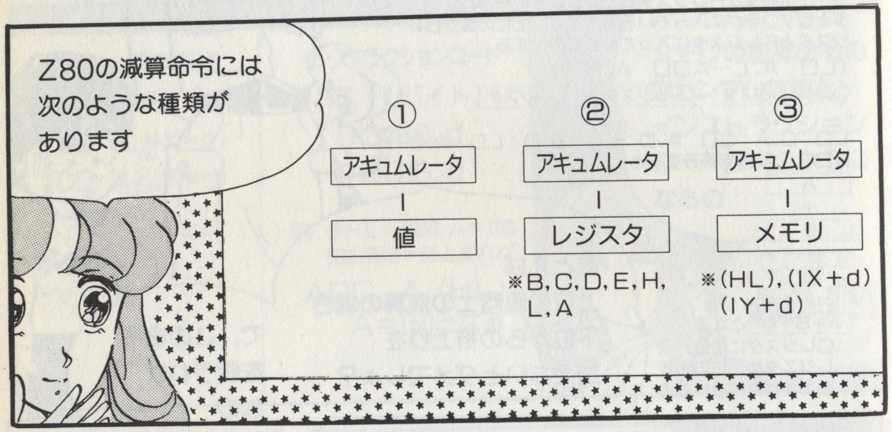


ねえ K子さん アキュムレータ同士の加算もできるの？
へえ～ なるへそ
そう！
これらの加算命令のアセンブリ言語は ADD A, rとなるの
ADD A, r
r=A,B,C,D,E,H,L
例 アキュムレータを2倍する
ADD A, A
これはちょっとしたテクニックでレジスタの内容を2倍にする時なんかに使うのよ
インストラクションコードはこのようになります
ADD A, r（rはレジスタ, A=A+r）
r=A,B,C,D,E,H,L
アセンブリ言語　インストラクションコード　動作
ADD A,B　80　A=A+B
ADD A,C　81　A=A+C
ADD A,D　82　A=A+D
ADD A,E　83　A=A+E
ADD A,H　84　A=A+H
ADD A,L　85　A=A+L
ADD A,A　87　A=A+A
フラグ
S ↕ Z ↕ P/V V Cy ↕
次はアキュムレータとメモリの加算命令を説明するね
アキュムレータ
+
メモリ
アキュムレータ + 値
ペアレジスタで間接的に指定されたアドレスの値
例 (HL)
こうするとHLレジスタの値で指定されたアドレスのメモリの値となる
この命令はアキュムレータにメモリに格納された値を加算するものなの


メモリのアドレスの指定方法は 次の3種類があります
HLレジスタで指定する方法
ADD A,(HL)
IXレジスタで指定する方法
ADD A,(IX+d)
dは8ビットのオフセットで最上位ビットが"1"の場合は負のオフセットとなる(−128〜0〜127)
IYレジスタで指定する方法
ADD A,(IY+d)
左と同じ
アセンブラ言語
インストラクションコード
ADD A,(HL)
86(1バイト)
ADD A,(IX+d)
DD,86,d
ADD A,(IY+d)
FD,86,d
フラグ
S↕ Z↕
P/V V C↕
例 今HL=5000, A=100
5000番地=50とすれば
ADD A,(HL)
A=150となる
アキュムレータとメモリの加算命令のアセンブリ言語とインストラクションコードは次のようになるの
ここまでわかった?
も、もち完璧!
信じて!!
うっうんちゃんとわかったよ
ふーん。そうえらいわねえ二人とも…
そんなによくわかったの
じゃあ……続いて特殊な加算命令の説明に入ることにしましょうか
墓穴…


特殊っていってもキャリーフラグもあわせて加算する点が違うだけなんだけどね
今までの命令
アキュムレータ + 値
今度の命令
アキュムレータ + 値 + Cy
まぜて
キャリー・フラグ?
そ!
ひとつのレジスタで演算できないような大きい値を扱うときのための桁上りの役目をもつフラグなの
ADC A,B
例 今,下位の値がHレジスタとLレジスタの和がAレジスタとCyフラグに入っているものとし,上位の値がDレジスタとEレジスタに入っているものとする
(LD A,L→ADD A,H)
Cy+ Aレジスタ ← Hレジスタ + Lレジスタ
A
LD C,A:LD A,D:LD B,D:LD A,E
ADC A,Bを実行すると
A
+
B
+
Cy
‖
A+B+Cy となる
∴Cレジスタに下位,Aレジスタに上位の値が演算されたことになる
ひっ…
ややこしい…
たとえば上位の値同士の演算の場合下位からの桁上りを加えないとダメでしょ?
で,この命令を使うわけなの
アセンブリ言語
インストラクションコード
ADC A,n CE,n(nは1バイトの値)
ADC A,B 88 A=A+B+Cy
ADC A,C 89 A=A+C+Cy
ADC A,D 8A A=A+D+Cy
ADC A,E 8B A=A+E+Cy
ADC A,H 8C A=A+H+Cy
ADC A,L 8D A=A+L+Cy
ADC A,A 8F A=A+A+Cy
ADC A,(HL) 8E A=A+(HL)+Cy
ADC A,(IX+d) DD,8E,d A=A(IX+d)+Cy
ADC A,(IY+d) FD,8E,d A=A(IY+d)+Cy
フラグ
S ↕ Z ↕ P/V V Cy ↕
この命令のアセンブリ言語とインストラクションコードは次のようになります





加算命令□はレジスタ，○はフラグ

オペランド

ADD $\boxed{\alpha}+\boxed{\beta}\rightarrow\boxed{\alpha}$

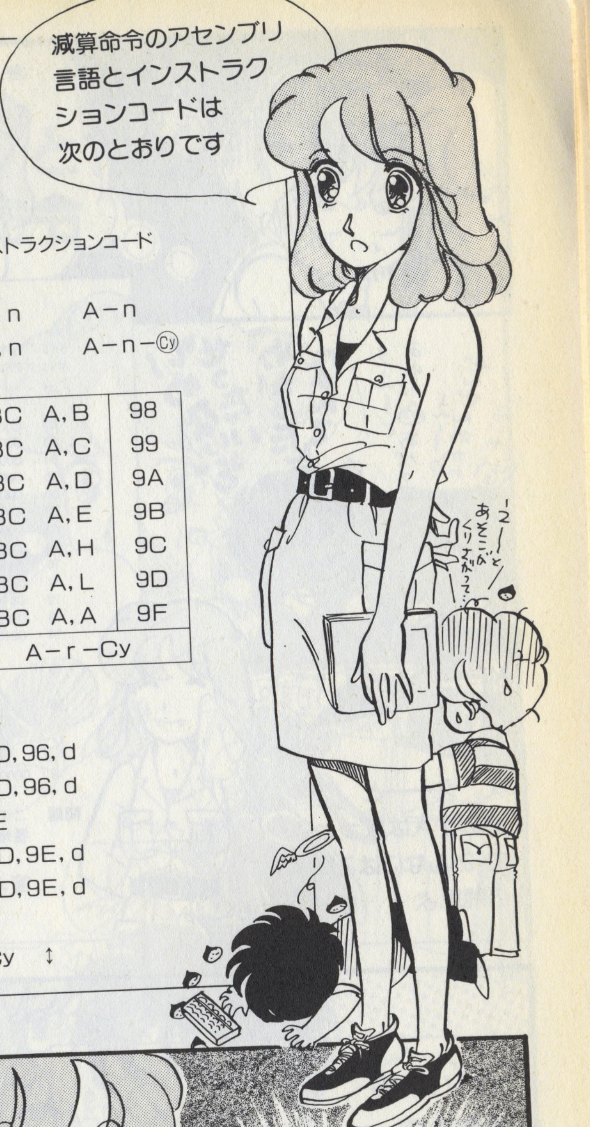
ADC $\boxed{\alpha}+\boxed{\beta}+\text{(Cy)}\rightarrow\boxed{\alpha}$

減算命令

SUB $\boxed{\alpha}-\boxed{\beta}\rightarrow\boxed{\alpha}$

SBC $\boxed{\alpha}-\boxed{\beta}-\text{(Cy)}\rightarrow\boxed{\alpha}$





減算命令

アセンブリ言語　　インストラクションコード

① アキュムレータと値

SUB n	D6, n	A−n
SBC A, n	DE, n	A−n−Cy

② アキュムレータとレジスタ

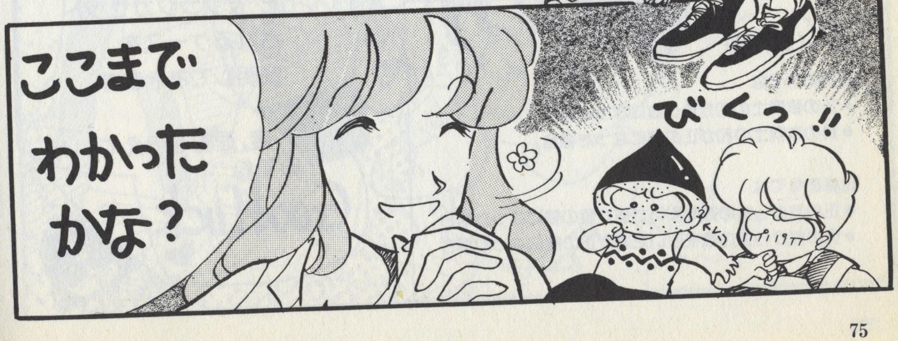
SUB B	90	SBC A, B	98
SUB C	91	SBC A, C	99
SUB D	92	SBC A, D	9A
SUB E	93	SBC A, E	9B
SUB H	94	SBC A, H	9C
SUB L	95	SBC A, L	9D
SUB A	97	SBC A, A	9F
A−r		A−r−Cy	

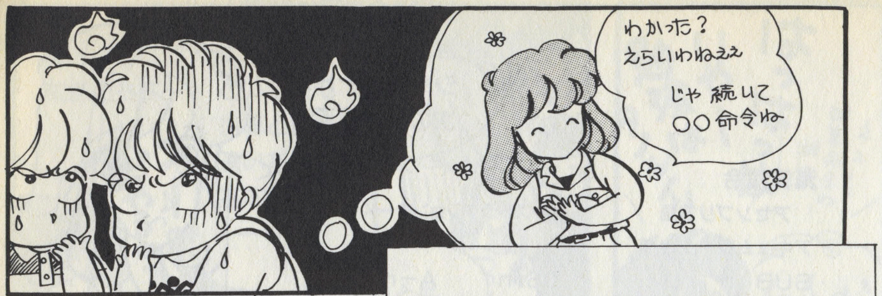
③ アキュムレータとメモリ

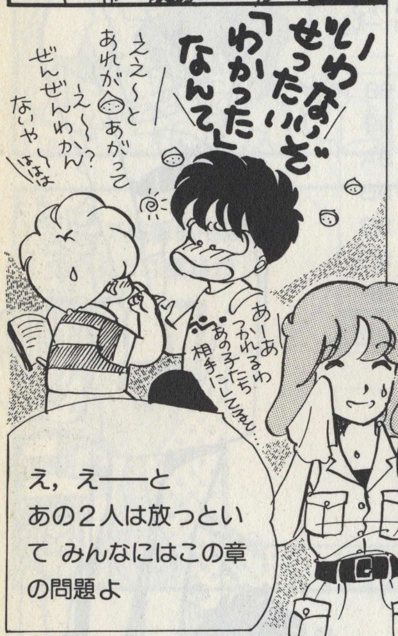
SUB (HL)	96
SUB (IX+d)	DD, 96, d
SUB (IY+d)	FD, 96, d
SBC A, (HL)	9E
SBC A, (IX+d)	DD, 9E, d
SBC A, (IY+d)	FD, 9E, d

フラグ

S ↕ Z ↕ P/V V Cy ↕







ライン ポインタ	メモリ アドレス	インストラクション コード	ラベル	オペコード	オペランド
01				ORG	0C000H
02	C000	3A00C1	START:	LD	A,(0C100H)
03	C003	3200D0		LD	(0D000H), A
04	C006	47		LD	B, A
05	C007	80		ADD	A, B
06	C008	3201D0		LD	(0D001H), A
07	C00B	4F		LD	C, A
08	C00C	87		ADD	A, A
09	C00D	3202D0		LD	(0D002H), A
10	C010	87		ADD	A, A
11	C011	3203D0		LD	(0D003H), A
12	C014	81		ADD	A, C
13	C015	3204D0		LD	(0D004H), A
14	C018	76		HALT	
15				ORG	0C100H
16	C100	0A		DB	10
17				ORG	0D00H
18	D000			DEFS	5

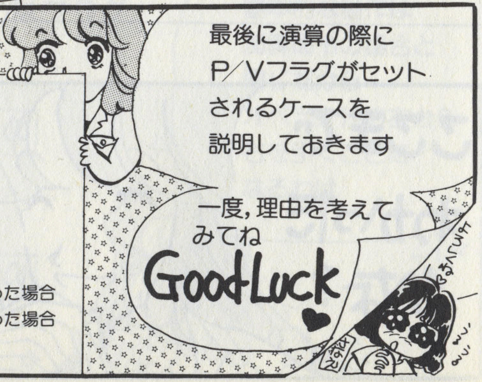
問題　このプログラムを実行すると，D000～D004番地の値はどうなるでしょう。

答えは78ページの欄外です。

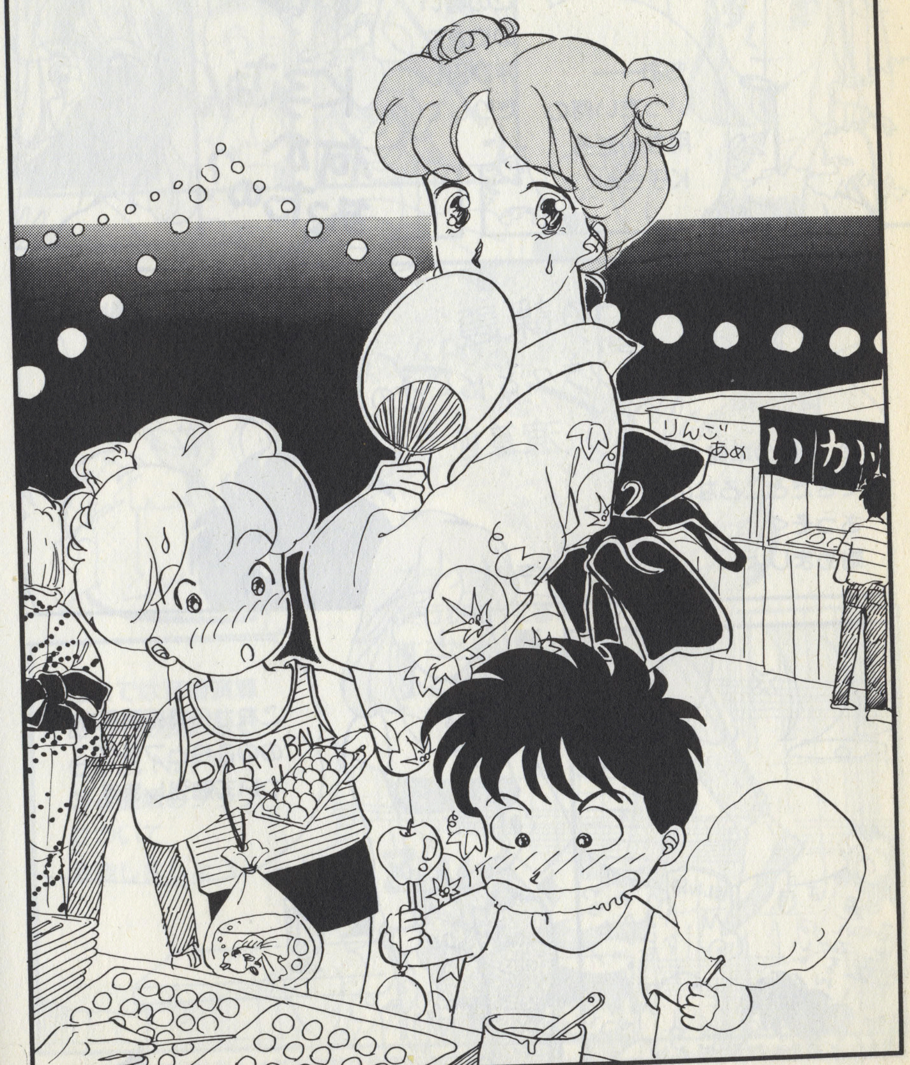
P/Vの内容

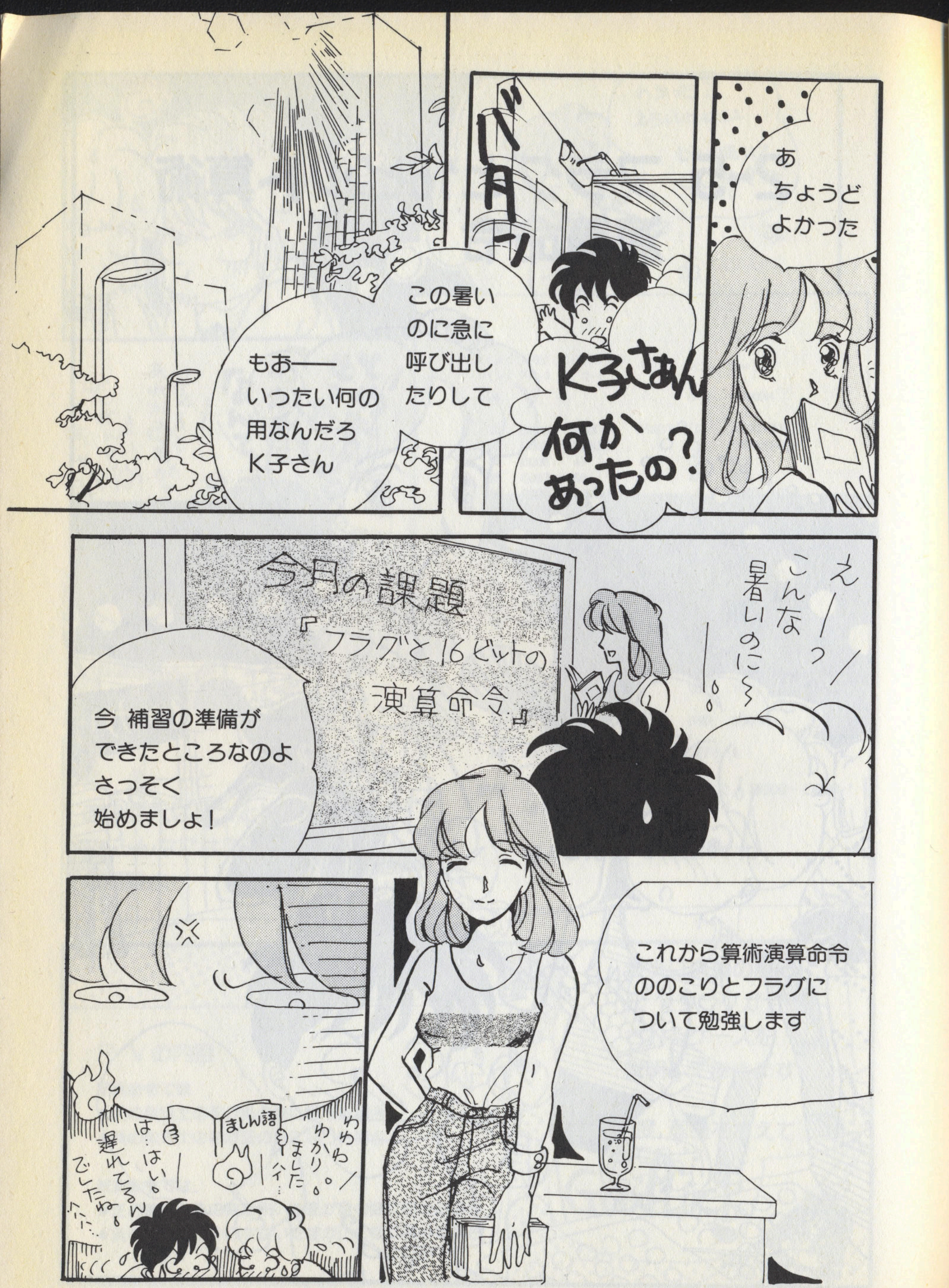
加算命令では
- 正の数同士の和が負の数になった場合
- 負の数同士の和が正の数になった場合

減算命令では
- 正の数から負の数を引いた差が負の数になった場合
- 負の数から正の数を引いた差が正の数になった場合

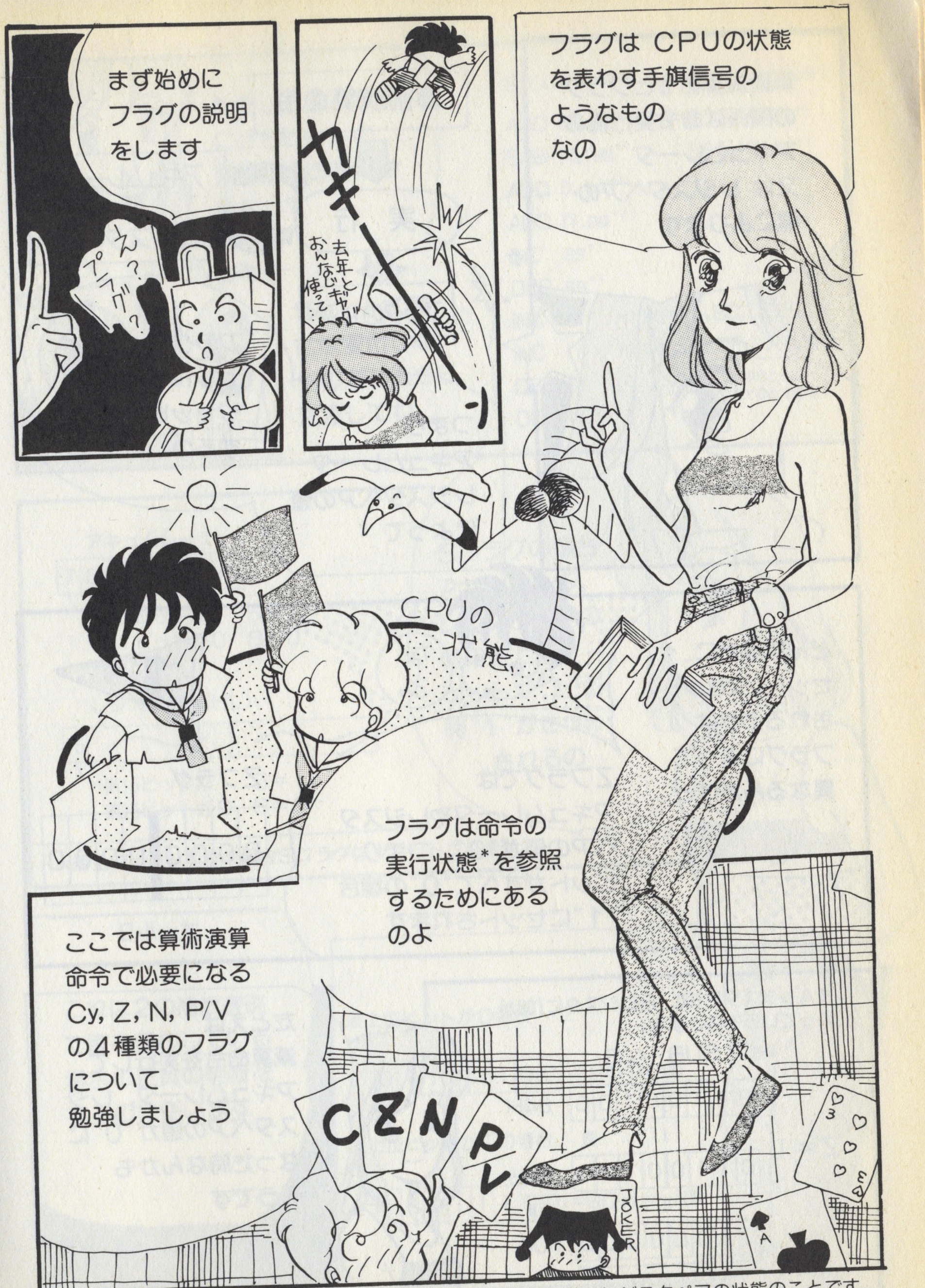


2-5 フラグと16ビット算術演算命令

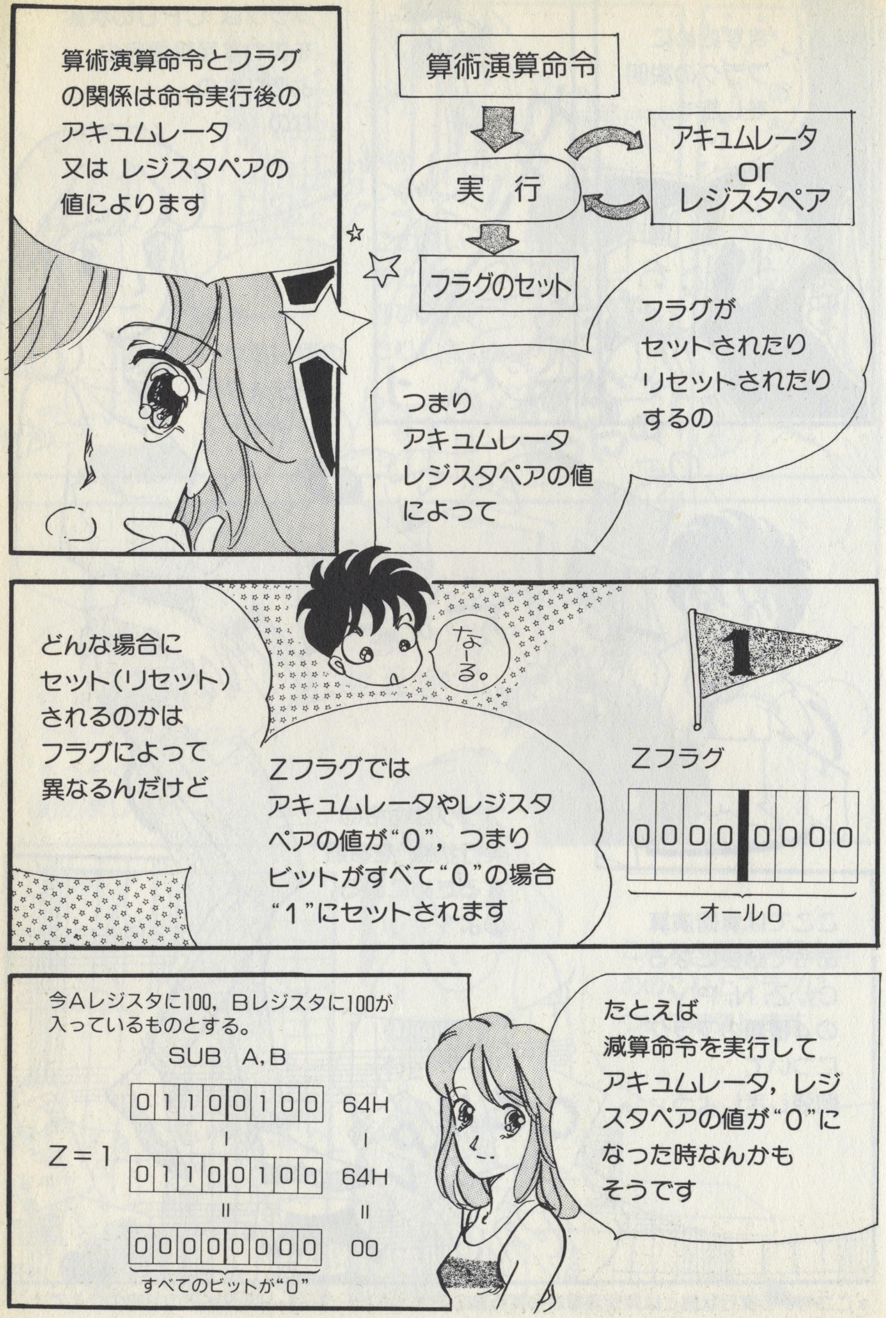


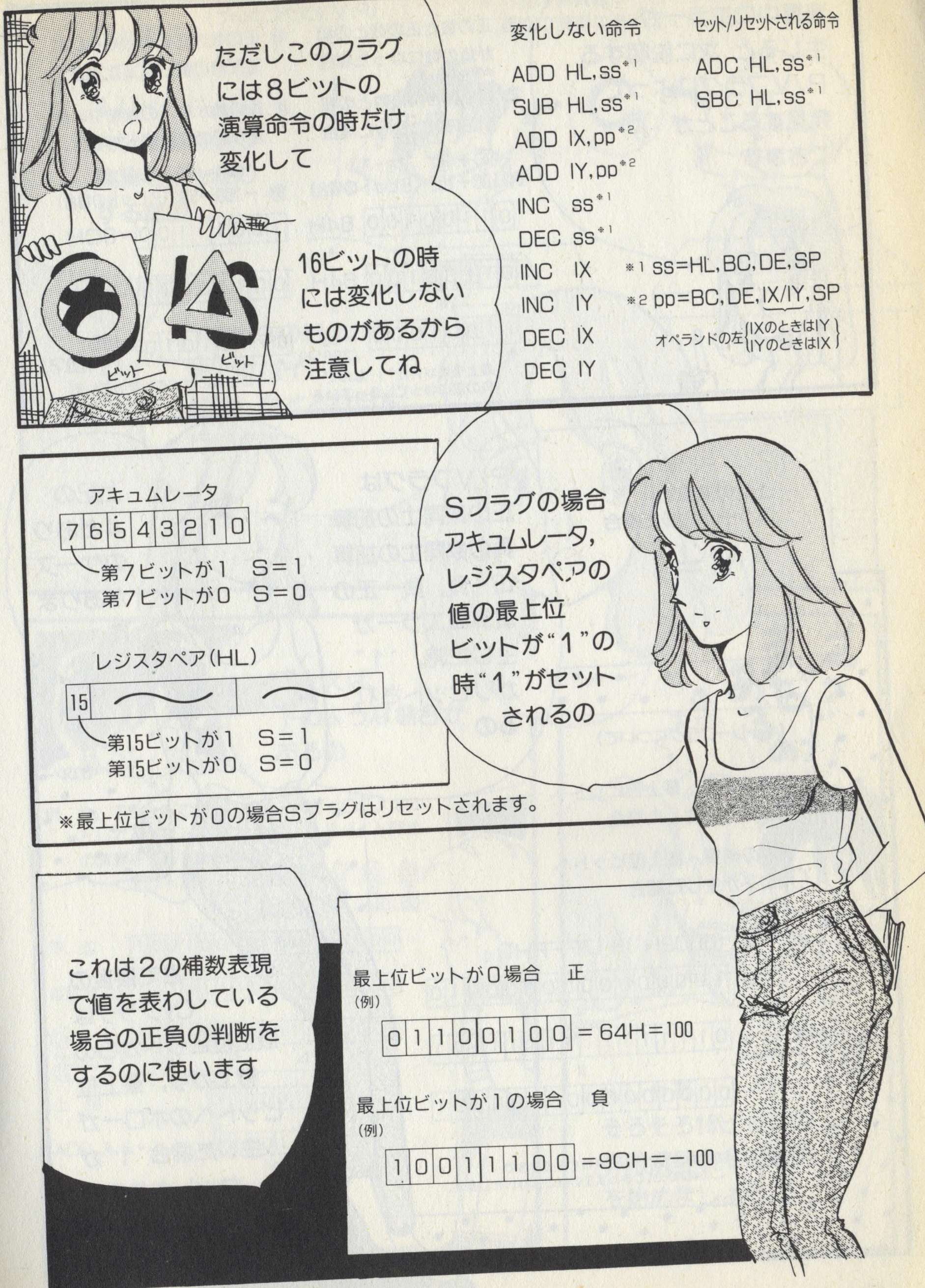


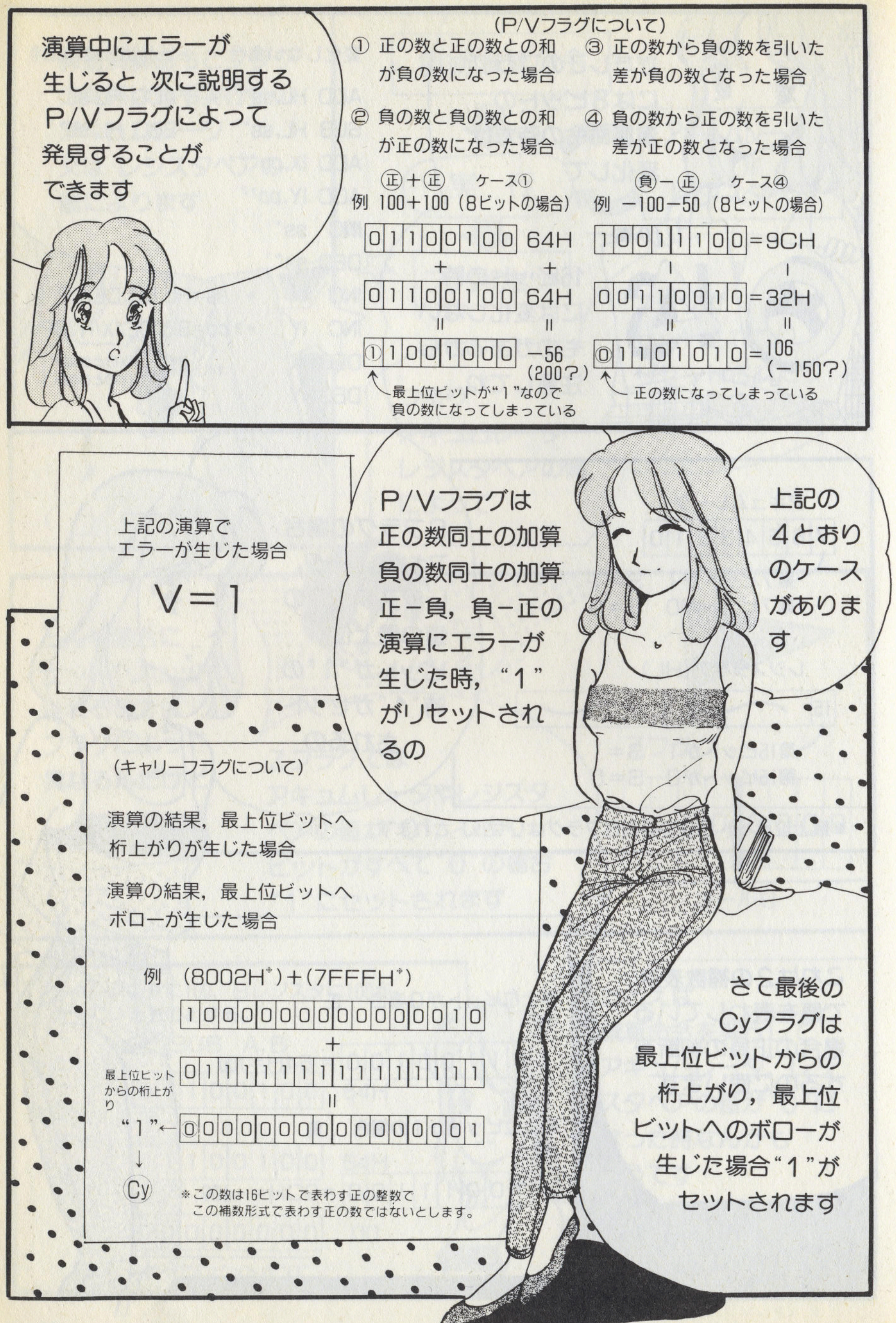
●76ページの答え ●0AH, 14H, 28H, 50H, 64H



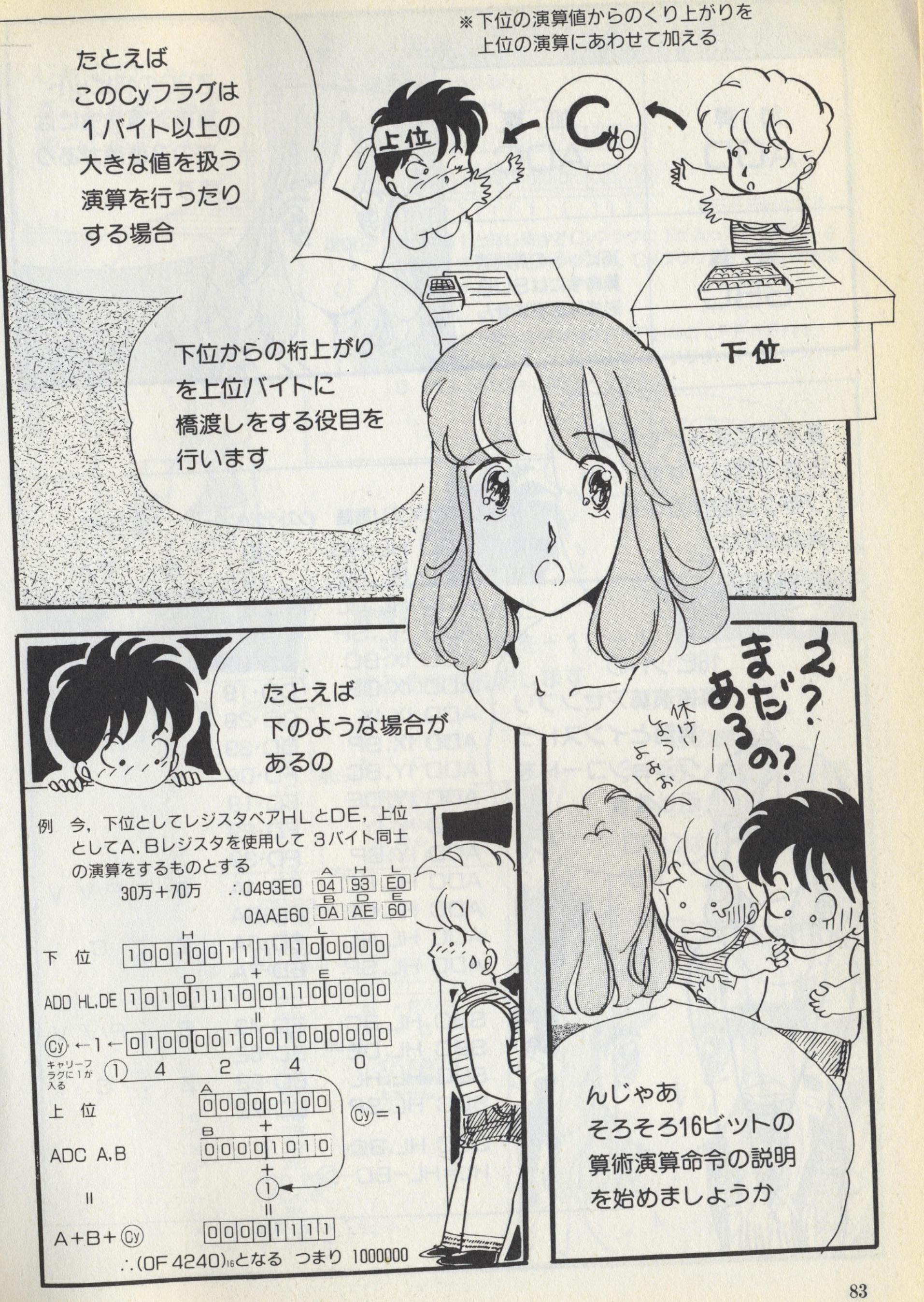
＊ここでいう実行状態とは算術演算命令実行後のアキュムレータ，レジスタペアの状態のことです。


算術演算命令とフラグの関係は命令実行後のアキュムレータ又は レジスタペアの値によります
算術演算命令
実 行
アキュムレータ or レジスタペア
フラグのセット
つまりアキュムレータレジスタペアの値によって
フラグがセットされたりリセットされたりするの
どんな場合にセット(リセット)されるのかはフラグによって異なるんだけど
な-る。
1
Zフラグではアキュムレータやレジスタペアの値が"0"，つまりビットがすべて"0"の場合"1"にセットされます
Zフラグ
0000 0000
オール0
今Aレジスタに100，Bレジスタに100が入っているものとする。
SUB A,B
01100100 64H
Z=1
01100100 64H
00000000 00
すべてのビットが"0"
たとえば減算命令を実行してアキュムレータ，レジスタペアの値が"0"になった時なんかもそうです


ただしこのフラグには8ビットの演算命令の時だけ変化して
16ビットの時には変化しないものがあるから注意してね
ビット
ビット
変化しない命令
ADD HL,ss＊1
SUB HL,ss＊1
ADD IX,pp＊2
ADD IY,pp＊2
INC ss＊1
DEC ss＊1
INC IX
INC IY
DEC IX
DEC IY
セット/リセットされる命令
ADC HL,ss＊1
SBC HL,ss＊1
＊1 ss=HL,BC,DE,SP
＊2 pp=BC,DE,IX/IY,SP
オペランドの左 {IXのときはIY IYのときはIX}
アキュムレータ
76543210
第7ビットが1 S=1
第7ビットが0 S=0
レジスタペア(HL)
15
第15ビットが1 S=1
第15ビットが0 S=0
※最上位ビットが0の場合Sフラグはリセットされます。
Sフラグの場合アキュムレータ，レジスタペアの値の最上位ビットが"1"の時"1"がセットされるの
これは2の補数表現で値を表わしている場合の正負の判断をするのに使います
最上位ビットが0場合 正
(例)
01100100 =64H=100
最上位ビットが1の場合 負
(例)
10011100 =9CH=−100

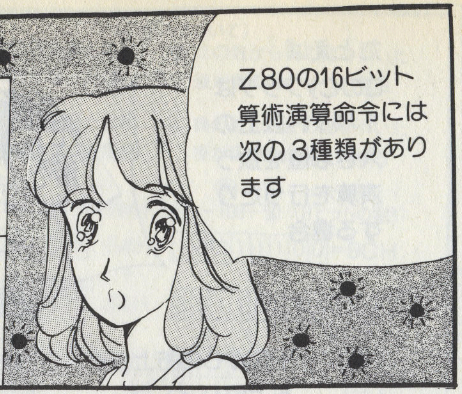


演算中にエラーが
生じると 次に説明する
P/Vフラグによって
発見することが
できます
(P/Vフラグについて)
① 正の数と正の数との和が負の数になった場合
② 負の数と負の数との和が正の数になった場合
③ 正の数から負の数を引いた差が負の数となった場合
④ 負の数から正の数を引いた差が正の数となった場合
㊣＋㊣ ケース①
例 100＋100 (8ビットの場合)
01100100 64H
＋ ＋
01100100 64H
＝ ＝
①1001000 −56 (200?)
最上位ビットが“1”なので負の数になってしまっている
負－㊣ ケース④
例 −100−50 (8ビットの場合)
10011100＝9CH
− −
00110010＝32H
＝ ＝
⓪1101010＝106 (−150?)
正の数になってしまっている
上記の演算でエラーが生じた場合
V＝1
上記の4とおりのケースがあります
P/Vフラグは正の数同士の加算 負の数同士の加算 正−負，負−正の演算にエラーが生じた時，“1”がリセットされるの
(キャリーフラグについて)
演算の結果，最上位ビットへ桁上がりが生じた場合
演算の結果，最上位ビットへボローが生じた場合
例 (8002H*)＋(7FFFH*)
1000000000000010
＋
0111111111111111
＝
“1”← 0000000000000001
最上位ビットからの桁上がり
↓
Cy
※この数は16ビットで表わす正の整数でこの補数形式で表わす正の数ではないとします。
さて最後のCyフラグは最上位ビットからの桁上がり，最上位ビットへのボローが生じた場合“1”がセットされます



※下位の演算値からのくり上がりを上位の演算にあわせて加える
たとえばこのCyフラグは1バイト以上の大きな値を扱う演算を行ったりする場合
上位
C
下位
下位からの桁上がりを上位バイトに橋渡しをする役目を行います
たとえば下のような場合があるの
例 今，下位としてレジスタペアHLとDE，上位としてA，Bレジスタを使用して 3バイト同士の演算をするものとする
30万＋70万 ∴0493E0 A 04 H 93 L E0
0AAE60 B 0A D AE E 60
下位 H 10010011 L 11100000
＋
ADD HL,DE D 10101110 E 01100000
＝
Cy ←1← 0100001001000000
キャリーフラグに1が入る ① 4 2 4 0
上位 A 00000100
＋
ADC A,B B 00001010
＋
①
＝ ＝
A＋B＋Cy 00001111
Cy＝1
∴(0F4240)16となる つまり 1000000
え?まだあるの?
休けいしようよぉぉ
んじゃあそろそろ16ビットの算術演算命令の説明を始めましょうか

加算	加算
ADD	ADC
減算 SBC	16ビットの算術演算命令にはSUB形式はありません





	アセンブリ言語	インストラクションコード	フラグ
加算命令	ADD HL,BC	09	S ・ P/V ・
	ADD HL,DE	19	Z ・ C ↕
	ADD HL,HL	29	
	ADD HL,SP	39	・変化しない
	ADD IX,BC	DD・09	↕ 演算結果にしたがってセット／リセットされる
	ADD IX,DE	DD・19	
	ADD IX,IX	DD・29	
	ADD IX,SP	DD・39	
	ADD IY,BC	FD・09	
	ADD IY,DE	FD・19	
	ADD IY,IY	FD・29	
	ADD IY,SP	FD・39	
	ADC HL,BC	ED・4A	S ↕ P/V V
	ADC HL,DE	ED・5A	
	ADC HL,HL	ED・6A	Z ↕ C ↕
	ADC HL,SP	ED・7A	
減算命令	SBC HL,BC	ED・42	S ↕ P/V V
	SBC HL,DE	ED・52	
	SBC HL,HL	ED・62	Z ↕ C ↕
	SBC HL,SP	ED・72	

動作　SBC HL,BC
HL=HL−BC−Ⓒy



問題1　今HLレジスタに7FFFH，BCレジスタに8000Hが入っているものとする。次の演算を実行するとHLレジスタの値はどうなるか。

ADD　HL,BC

答

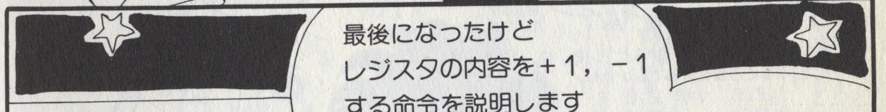
```
  0 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1
+
  1 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0
---------------------------------------
  1 1 1 1 1 1 1 1   1 1 1 1 1 1 1 1     ∴FFFFHとなる
```

問題2　今，問題1と同じ条件でCyフラグに1が入っていた場合，次の演算を実行するとHレジスタ，Cyフラグ，Zフラグの値はどうなるか。

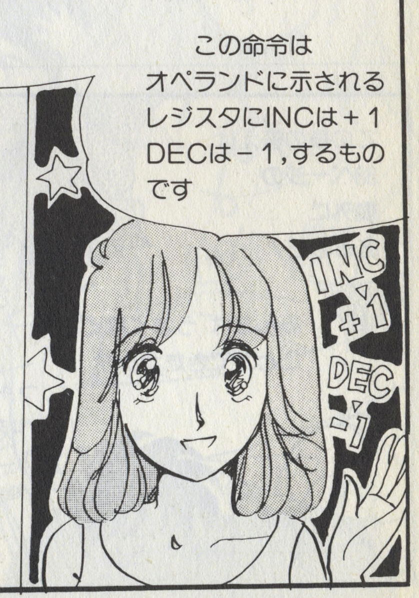
ADC　HL,BC

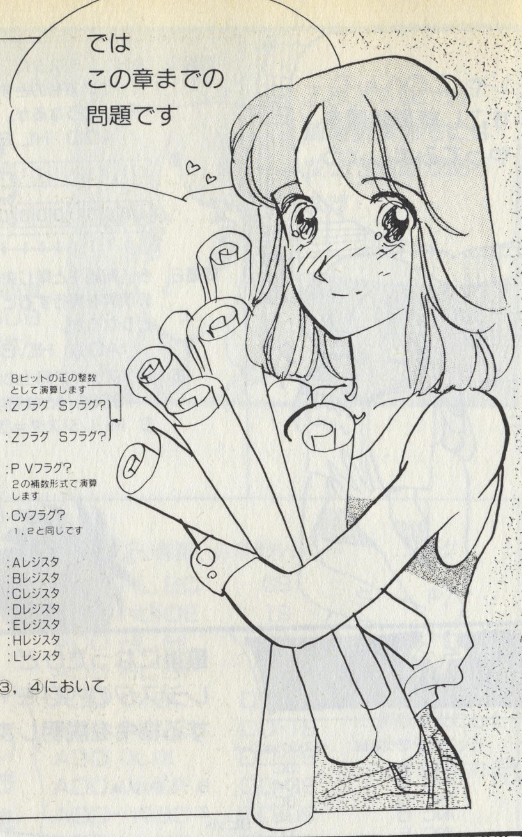
答　レジスタペア同士の演算はFFFFHなのでFFFFH+1=10000Hとなる。したがって Cyフラグ=1 Zフラグ=0 HLレジスタ=0000H となる





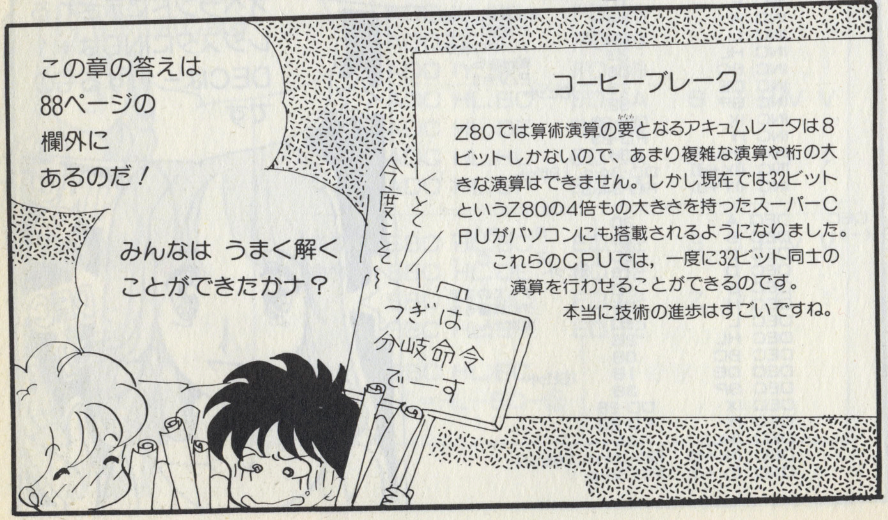
	アセンブリ言語	インストラクションコード		フラグ
INC (+1)	INC A	3C	8ビット	S ↕ P/V ✳
	INC B	04		Z ↕ C ・
	INC C	0C		
	INC D	14		✳演算を行う前に指定したレジスタの値が80HならP/V=1
	INC E	1C		
	INC H	24		
	INC L	2C		
	INC HL	23	16ビット	
	INC BC	03		
	INC DE	13		
	INC SP	33		
	INC IX	DD・23		
	INC IY	FD・23		
	INC (HL)	34	8ビット	
	INC (IX+d)	DD・34・d		
	INC (IY+d)	FD・34・d		
DEC (−1)	DEC A	3D	8ビット	S ・ P/V ・
	DEC B	05		Z ・ C ・
	DEC C	0D		
	DEC D	15		すべてのフラグに影響を与えません
	DEC E	1D		
	DEC H	25		
	DEC L	2D		
	DEC HL	2B	16ビット	
	DEC BC	0B		
	DEC DE	1B		
	DEC SP	3B		
	DEC IX	DD・2B		
	DEC IY	FD・2B		
	DEC (HL)	35	8ビット	
	DEC (IX+d)	DD・35・d		
	DEC (IY+d)	FD・35・d		






```
メモリ  インストラクション  ラベル   オペコード  オペランド
アドレス コード
                                ORG       0C000H
C000  3A06C1      START:  LD        A,(0C106H)
C003  6F                  LD        L,A
C004  3A05C1              LD        A,(0C105H)
C007  67                  LD        H,A
C008  3A04C1              LD        A,(0C104H)
C00B  5F                  LD        E,A
C00C  3A03C1              LD        A,(0C103H)
C00F  57                  LD        D,A
C010  3A02C1              LD        A,(0C102H)
C013  4F                  LD        C,A
C014  3A01C1              LD        A,(0C101H)
C017  47                  LD        B,A         8ビットの正の整数
C018  3A00C1              LD        A,(0C100H)  として演算します
C01B  80                  ADD       A, B        ;Zフラグ  Sフラグ?
C01C  79                  LD        A,C
C01D  92                  SUB       D           ;Zフラグ  Sフラグ?
C01E  7B                  LD        A,E
C01F  84                  ADD       A,H         ;P Vフラグ?
C020  7B                  LD        A,E         2の補数形式で演算
C021  85                  ADD       A,L         します
C022  84                  ADD       A,H         ;Cyフラグ?
C023  76                  HALT                  1,2と同じです
                          ORG       0C100H
C100  64                  DB        100         ;Aレジスタ
C101  64                  DB        100         ;Bレジスタ
C102  4D                  DB        77          ;Cレジスタ
C103  4D                  DB        77          ;Dレジスタ
C104  7F                  DB        127         ;Eレジスタ
C105  01                  DB        1           ;Hレジスタ
C106  80                  DB        128         ;Lレジスタ
```

問題　このプログラムを実行すると①，②，③，④において
フラグはどう変化するでしょうか。
① Z，S　② Z，S
③ P/V　④ Cy



コーヒーブレーク

Z80では算術演算の要となるアキュムレータは8ビットしかないので，あまり複雑な演算や桁の大きな演算はできません。しかし現在では32ビットというZ80の4倍もの大きさを持ったスーパーCPUがパソコンにも搭載されるようになりました。これらのCPUでは，一度に32ビット同士の演算を行わせることができるのです。本当に技術の進歩はすごいですね。

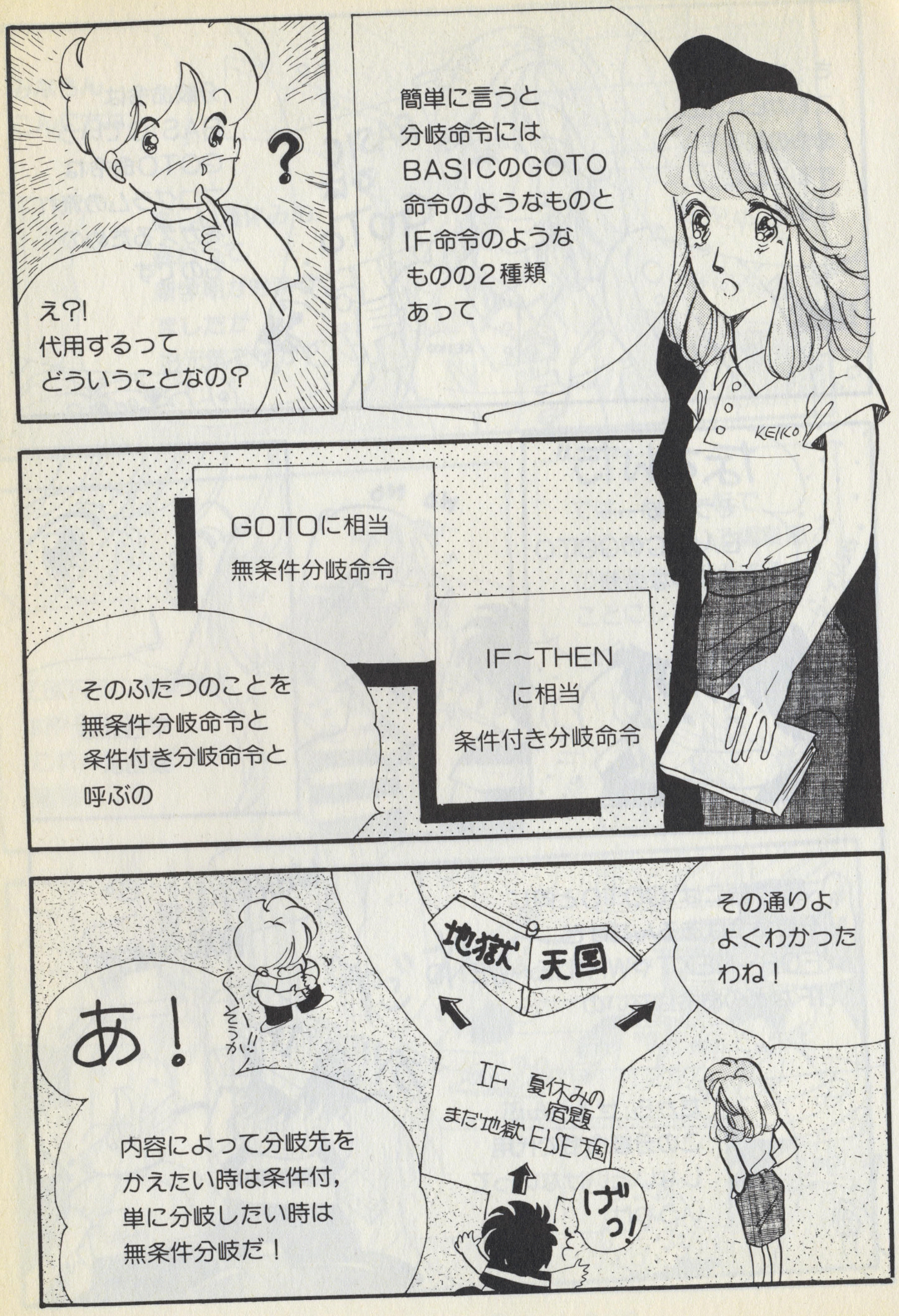
2-6 Z80の分岐命令

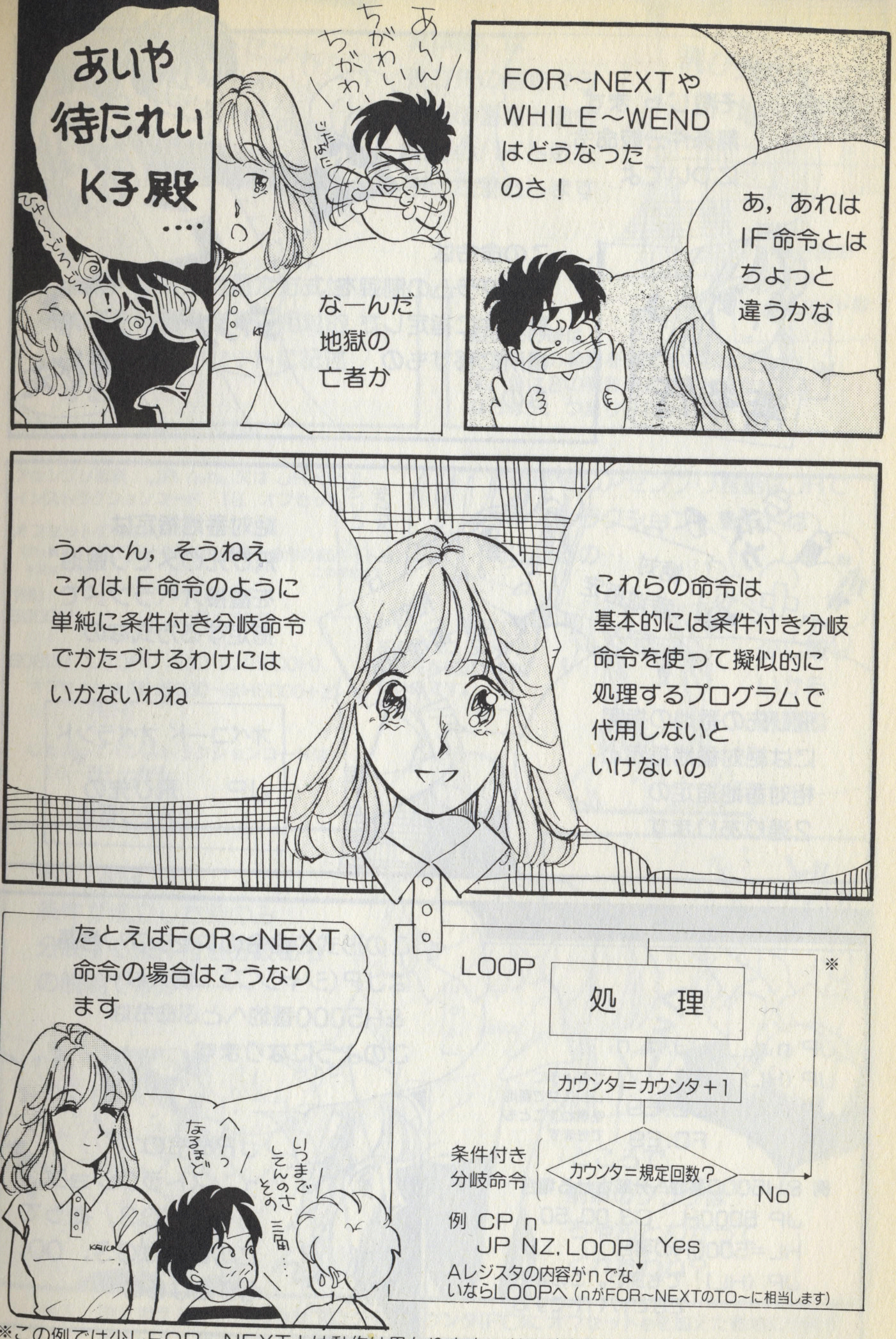




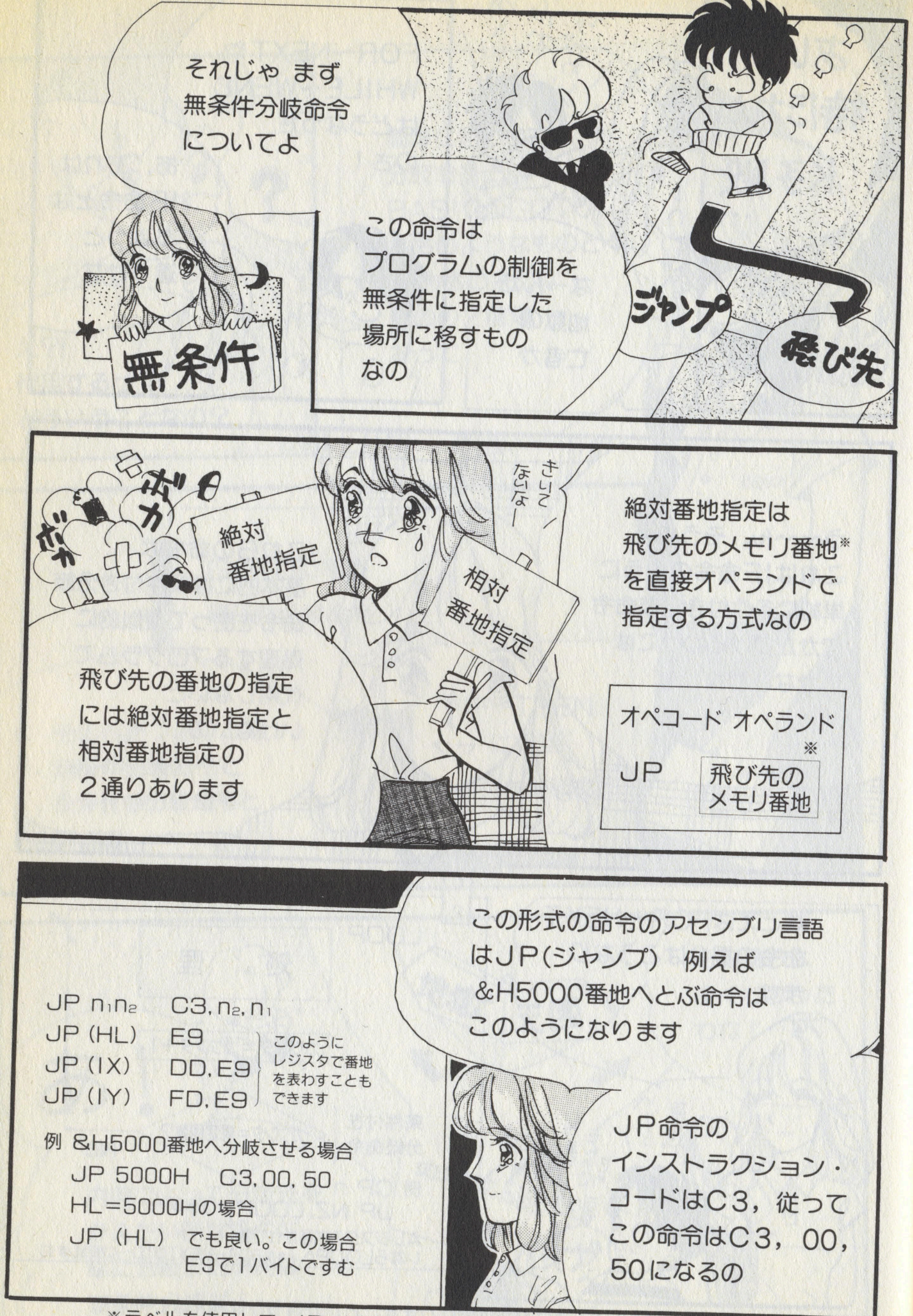


●86ページの答え● ①Z＝0, S＝1 ②Z＝1, S＝0
③P/V＝1 ④Cy＝1

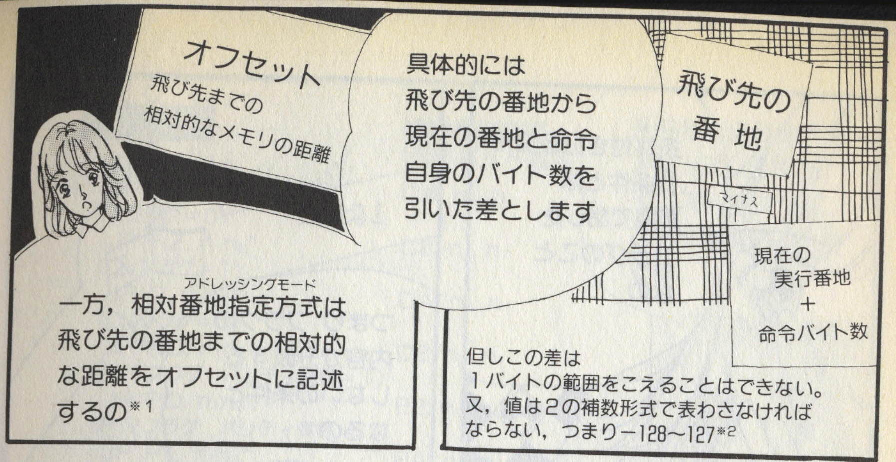




※この例では少しFOR～NEXTとは動作は異なります。どこが違うかはみんなで考えてみてください。



※ラベルを使用して JP KEIKO とすることもできます。



アセンブリ言語 ：JR n_1n_2 又は JR ラベル
インストラクションコード 18，オフセット

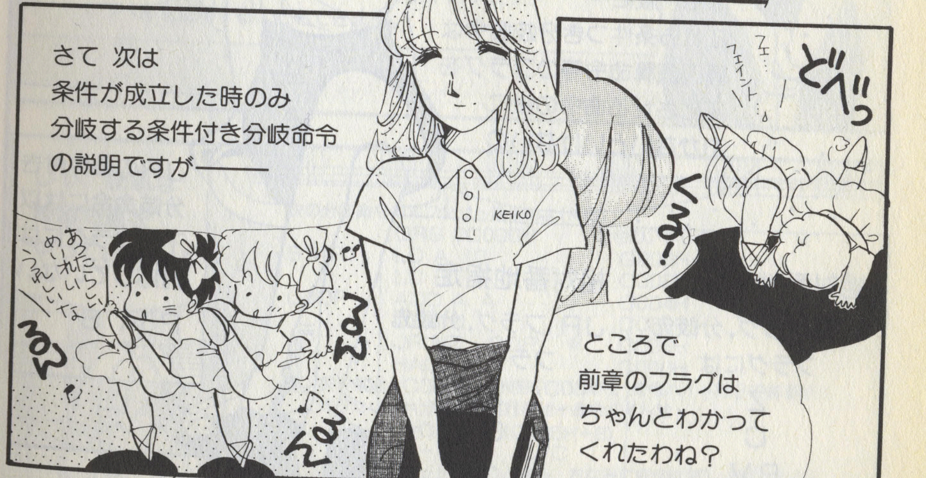
オフセットの求め方
（n_1n_2※－現在実行中の命令番地＋命令の総バイト数）
※又はラベル　　　　この場合は2

【例】
5000H　　　　　　JR LOOP
　　　　　　　　　　～
5080H　LOOP:　LD A, (6000H)
オフセットは(&H5080－&H5000＋2)
∴ 7EH

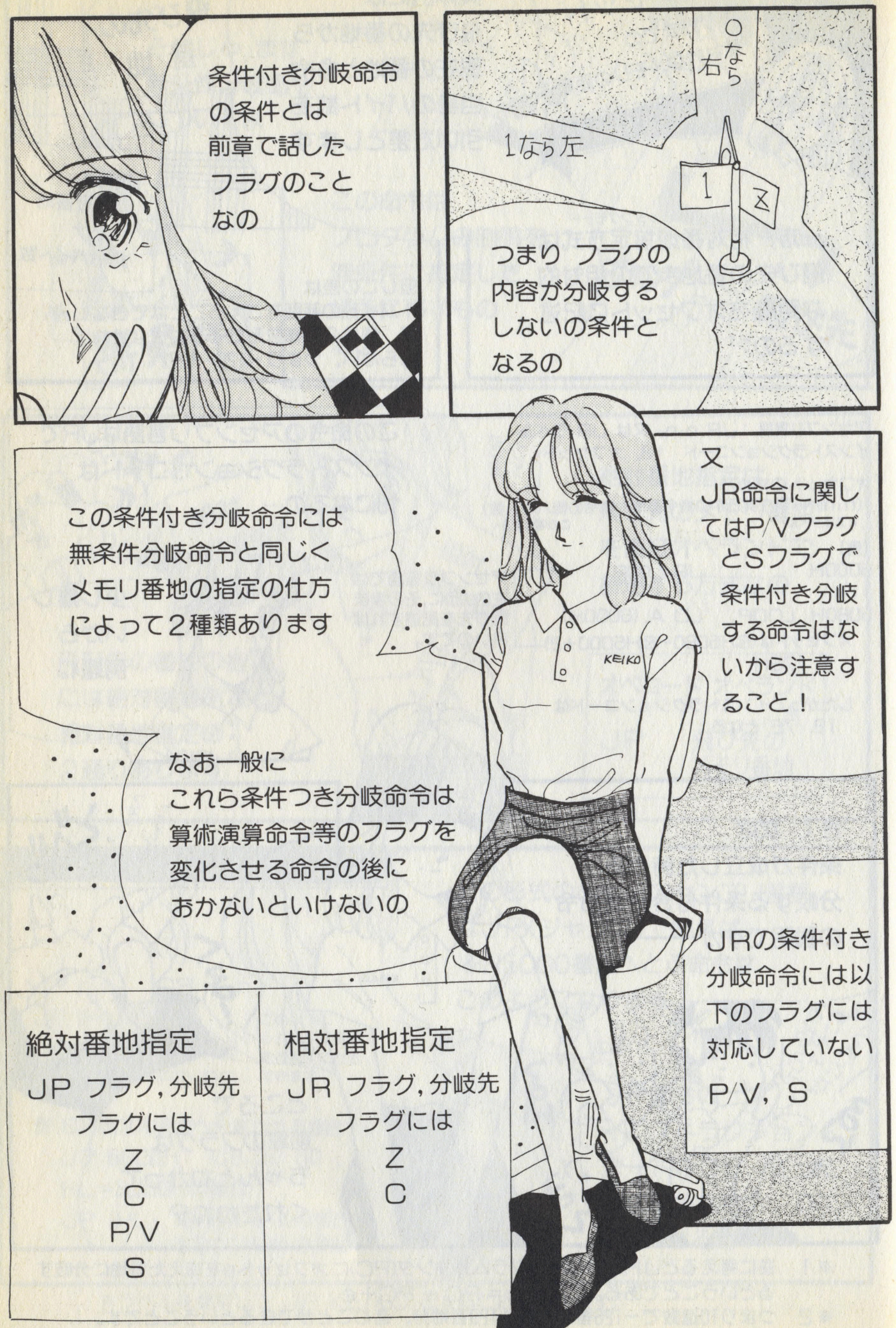
アセンブリ言語では左のように，そのまま飛び先を記述すれば良いのです。

したがってインストラクションコードは
18，7E となる

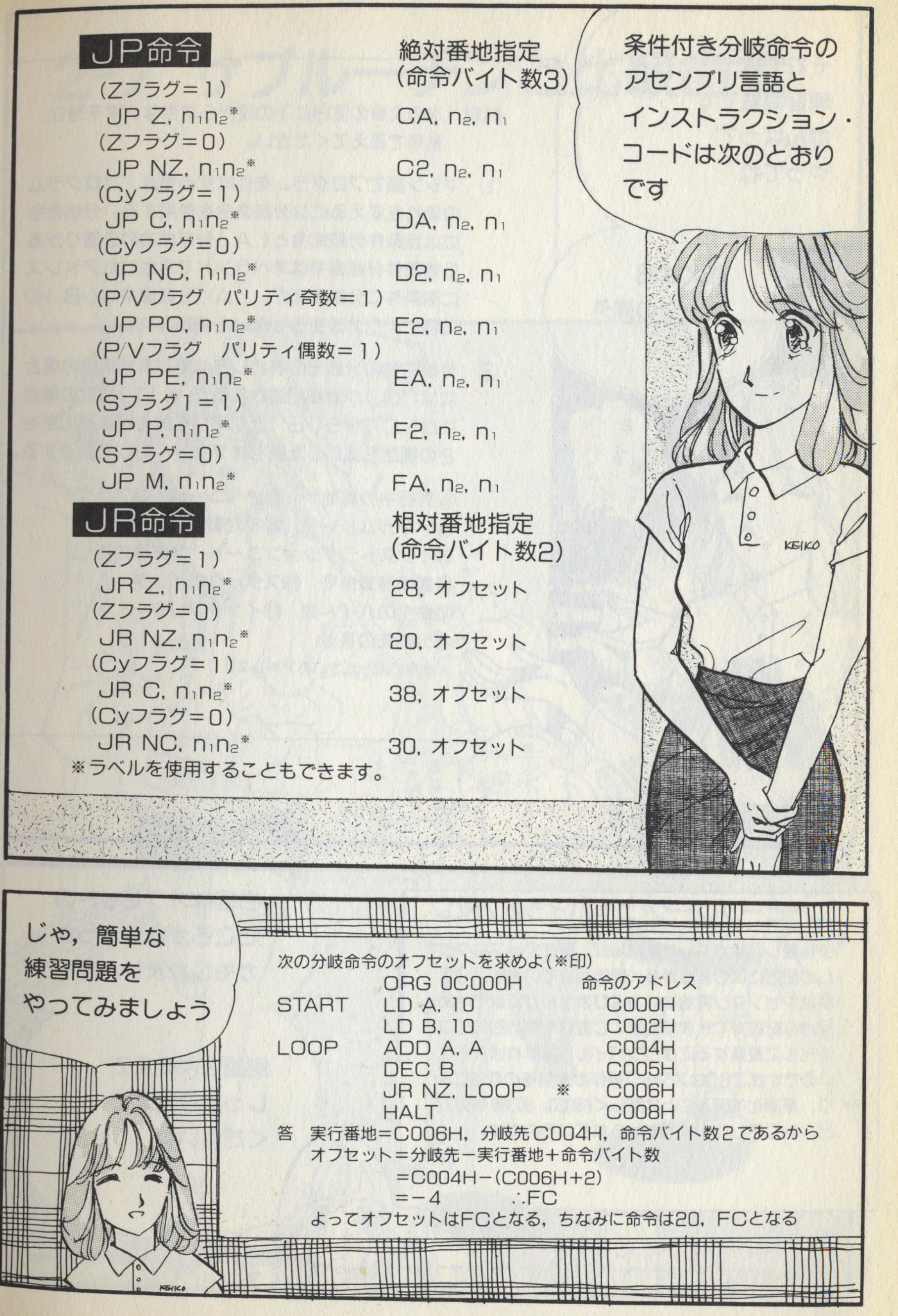




※1　逆に考えるとJR命令はプログラムカウンタ(PC)にオフセットeを加えた番地に分岐するということである。∴18, e＝PC←PC＋e
※2　つまり10進数で－126番地から＋129番地分，進むことができるということです。(現在の番地から数えて)



条件付き分岐命令の条件とは前章で話したフラグのことなの
○なら右
1なら左
つまり フラグの内容が分岐するしないの条件となるの
この条件付き分岐命令には無条件分岐命令と同じくメモリ番地の指定の仕方によって2種類あります
又JR命令に関してはP/VフラグとSフラグで条件付き分岐する命令はないから注意すること！
なお一般にこれら条件つき分岐命令は算術演算命令等のフラグを変化させる命令の後におかないといけないの
絶対番地指定
JP フラグ，分岐先
フラグには
Z
C
P/V
S
相対番地指定
JR フラグ，分岐先
フラグには
Z
C
JRの条件付き分岐命令には以下のフラグには対応していない
P/V，S



JP命令
絶対番地指定（命令バイト数3）
（Zフラグ＝1）
JP Z, n_1n_2* CA, n_2, n_1
（Zフラグ＝0）
JP NZ, n_1n_2* C2, n_2, n_1
（Cyフラグ＝1）
JP C, n_1n_2* DA, n_2, n_1
（Cyフラグ＝0）
JP NC, n_1n_2* D2, n_2, n_1
（P/Vフラグ パリティ奇数＝1）
JP PO, n_1n_2* E2, n_2, n_1
（P/Vフラグ パリティ偶数＝1）
JP PE, n_1n_2* EA, n_2, n_1
（Sフラグ1＝1）
JP P, n_1n_2* F2, n_2, n_1
（Sフラグ＝0）
JP M, n_1n_2* FA, n_2, n_1
JR命令
相対番地指定（命令バイト数2）
（Zフラグ＝1）
JR Z, n_1n_2* 28, オフセット
（Zフラグ＝0）
JR NZ, n_1n_2* 20, オフセット
（Cyフラグ＝1）
JR C, n_1n_2* 38, オフセット
（Cyフラグ＝0）
JR NC, n_1n_2* 30, オフセット
※ラベルを使用することもできます。
条件付き分岐命令のアセンブリ言語とインストラクション・コードは次のとおりです
じゃ，簡単な練習問題をやってみましょう
次の分岐命令のオフセットを求めよ（※印）
ORG 0C000H 命令のアドレス
START LD A, 10 C000H
LD B, 10 C002H
LOOP ADD A, A C004H
DEC B C005H
JR NZ, LOOP ※ C006H
HALT C008H
答 実行番地＝C006H，分岐先C004H，命令バイト数2であるから
オフセット＝分岐先－実行番地＋命令バイト数
＝C004H－（C006H＋2）
＝－4 ∴FC
よってオフセットはFCとなる，ちなみに命令は20，FCとなる



問題　次の文章の空白に下の段から適当な文字を選び，番号で答えてください。

⑴　マシン語でプログラムを作成する場合，プログラムの流れを変えるには分岐命令を使用する。分岐命令には無条件分岐命令と（ A ）分岐命令の2通りがあり無条件分岐命令はオペランドで示されたアドレスに無条件に分岐するが，（ A ）分岐命令は（ B ）の内容により分岐するしないが決定される。

⑵　分岐命令の分岐先の表わし方は絶対番地指定の場合には（ C ）が直接記述されるが，（ D ）指定の場合には（ C ）から（ E ）と（ F ）を加えた値との差を2の補数形式にした値をオフセットとして記述する。

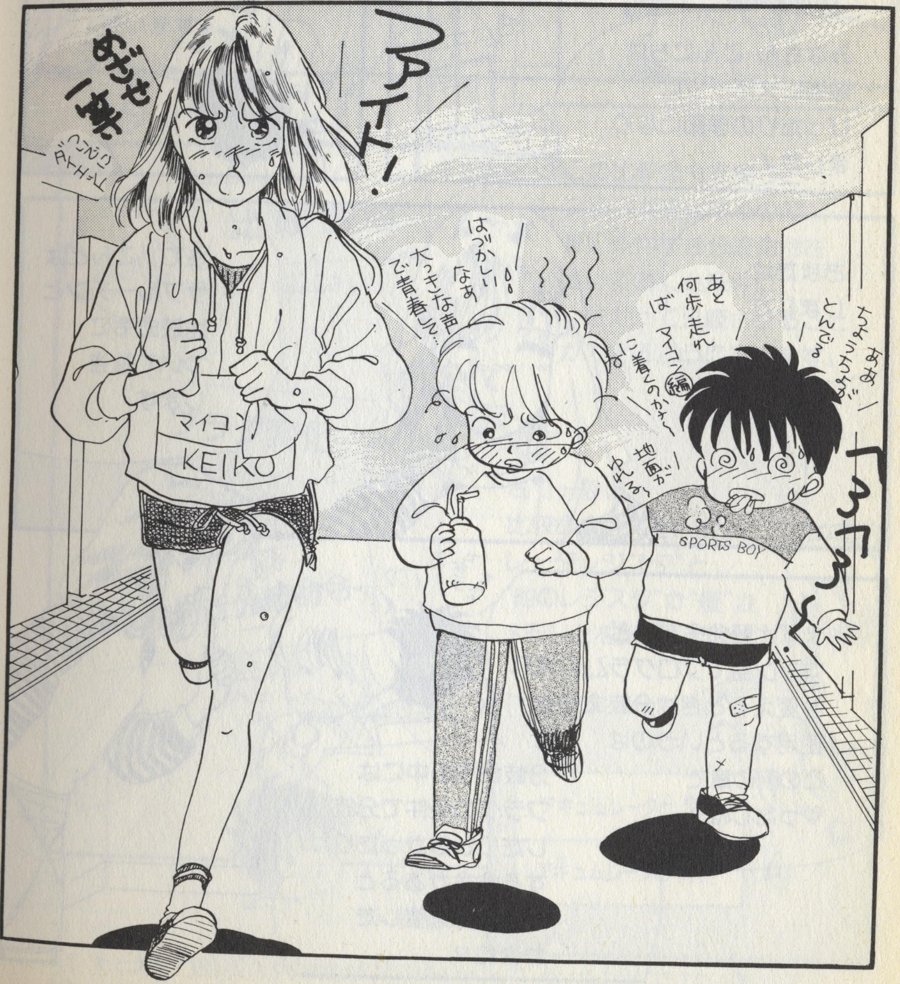
①実行中の番地※　②オペコード　③フラグ
④アキュムレータ　⑤相対番地
⑥インストラクションコード　⑦条件付き
⑧算術演算命令　⑨スタックポインタ
⑩命令のバイト数　⑪インタプリタ
⑫分岐先の番地（アドレス）
※命令の置かれているアドレス

コーヒーブレーク『コンパイラとZ80』

少し難しい話かもしれませんが，現在ではプログラムの記述にはC言語が多く使用されています。C言語はアセンブリ言語にはおよびませんが柔軟できめ細かな記述ができます。さてC言語を実行形式のファイルに変換するにはコンパイルしなければならないのですが，Z80はスタック操作が貧弱等の理由により，最適化を図ることは難しく68020，80386等の32ビットCPUに期待がかけられているのです。



2-7 サブルーチンと比較命令

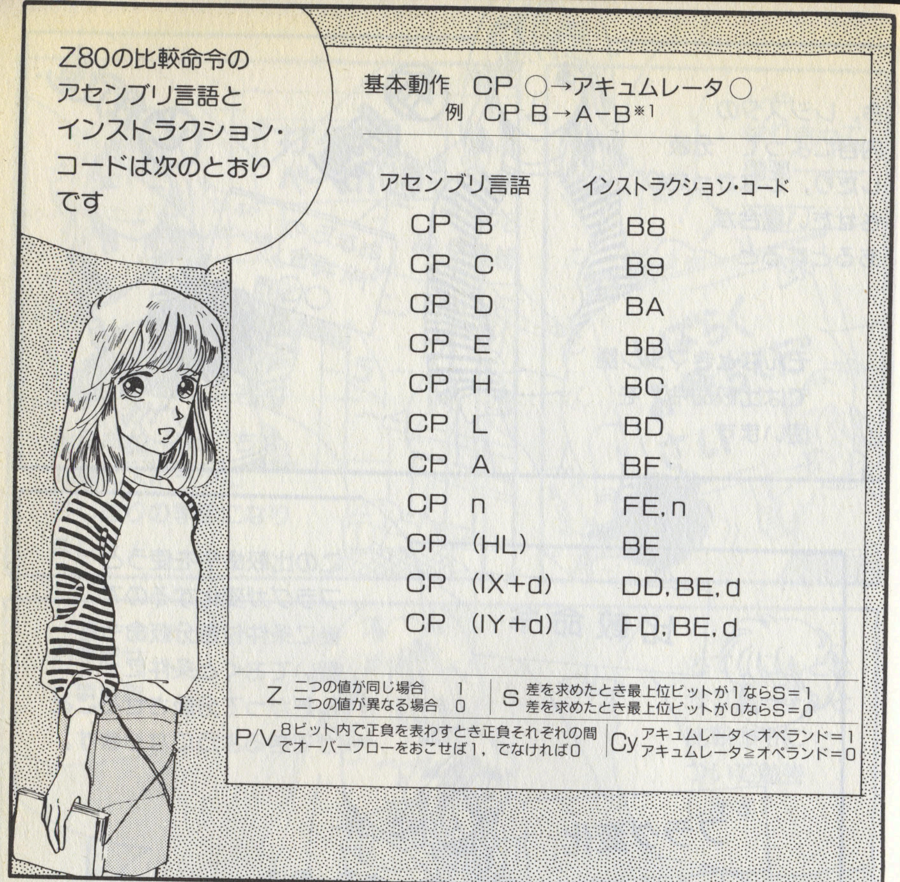




●96ページの答え ●A-7, B-3, C-12, D-5, E-1, F-10



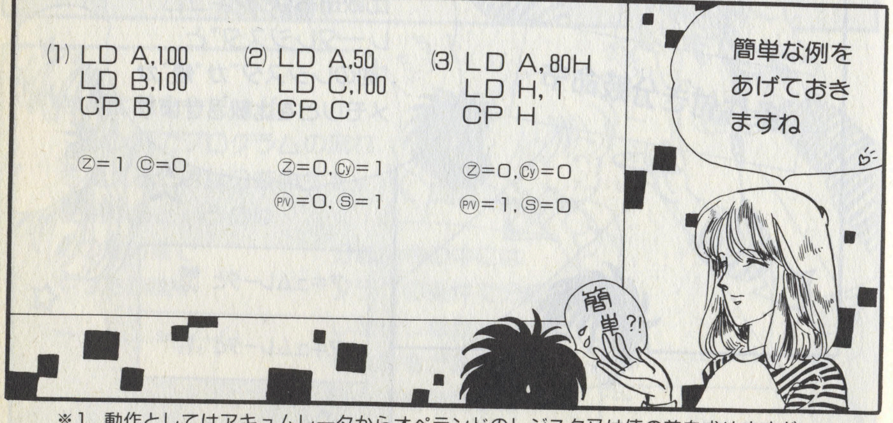
※1 たとえば "Bレジスタの内容が〇〇ならジャンプ" というようにさせたい場合など。
※2 メモリとの比較は直接，メモリ番地で指定できません。レジスタを使って間接的に指定します。



基本動作　CP ◯→アキュムレータ◯
例　CP B→A−B※1

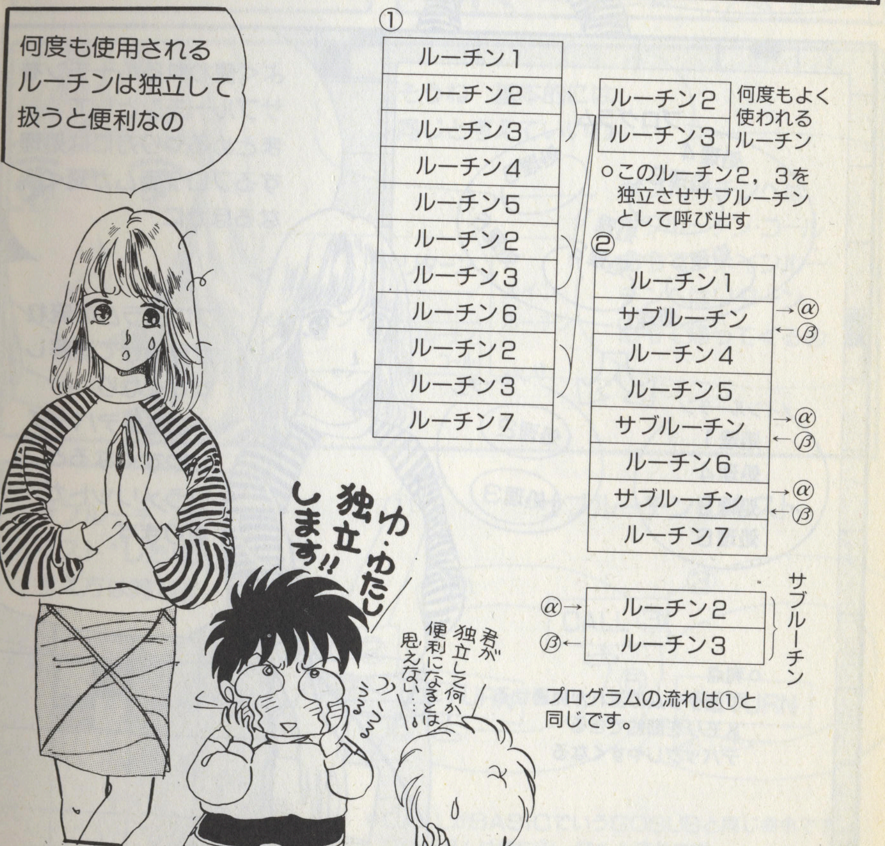
アセンブリ言語	インストラクション・コード
CP B	B8
CP C	B9
CP D	BA
CP E	BB
CP H	BC
CP L	BD
CP A	BF
CP n	FE, n
CP (HL)	BE
CP (IX+d)	DD, BE, d
CP (IY+d)	FD, BE, d

Z	二つの値が同じ場合　1 二つの値が異なる場合　0	S	差を求めたとき最上位ビットが1ならS=1 差を求めたとき最上位ビットが0ならS=0
P/V	8ビット内で正負を表わすとき正負それぞれの間でオーバーフローをおこせば1，でなければ0	Cy	アキュムレータ<オペランド=1 アキュムレータ≧オペランド=0

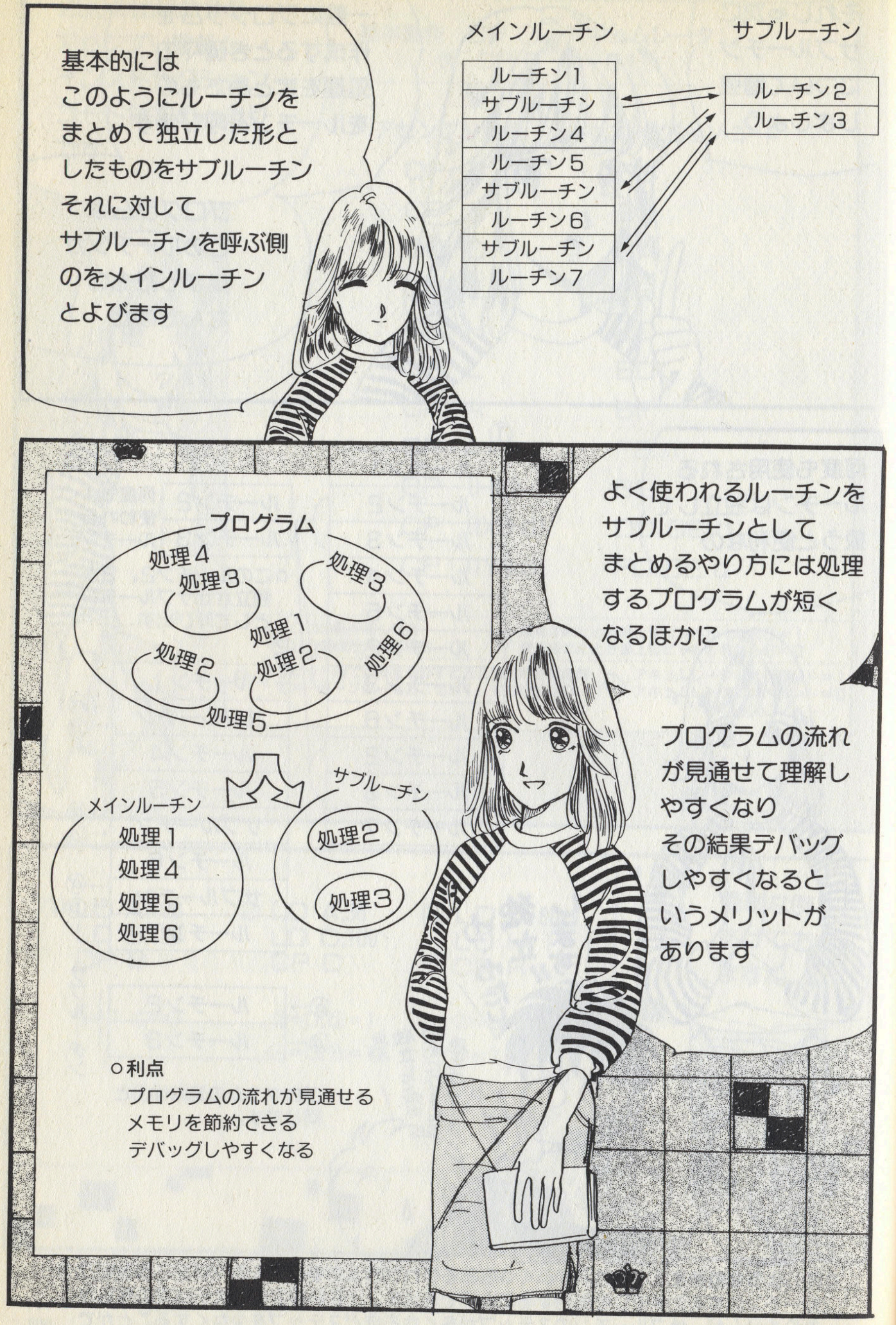


※1 動作としてはアキュムレータからオペランドのレジスタ又は値の差を求めますが，アキュムレータ自身の値は変化しません。変化するのはフラグのみです。



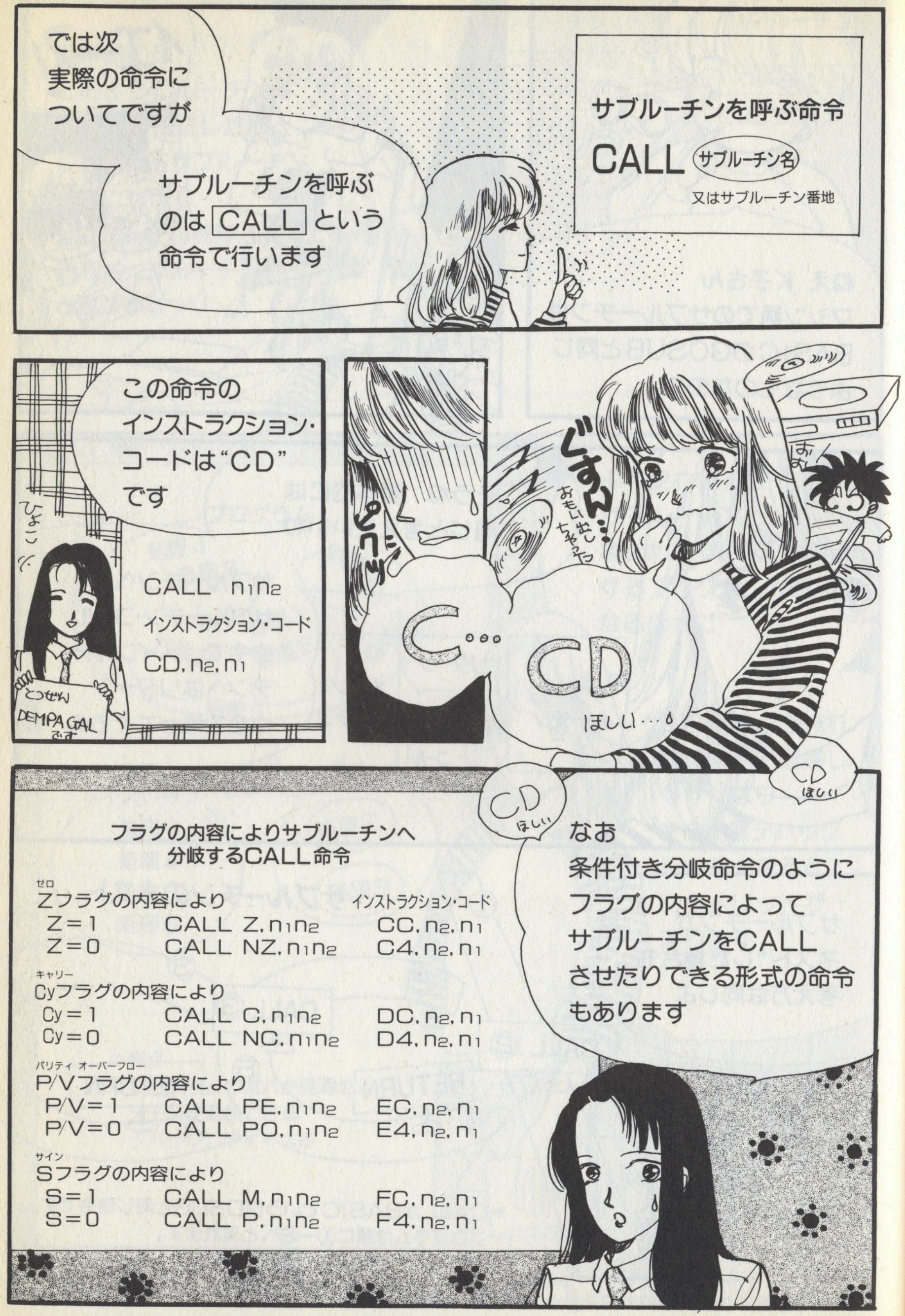


○同じような処理をするところがいくつもあるプログラムの場合，その同じところをサブルーチンにするとメモリを節約することができます。上記の例では1ステップの差しかありませんが，サブルーチンのステップが多くなるほどステップを少なくすることができます。





※サブルーチンの中にサブルーチンがあるような状態。



フラグの内容によりサブルーチンへ分岐するCALL命令

		インストラクション・コード
Zフラグの内容により		
Z=1	CALL Z, n_1n_2	CC, n_2, n_1
Z=0	CALL NZ, n_1n_2	C4, n_2, n_1
Cyフラグの内容により		
Cy=1	CALL C, n_1n_2	DC, n_2, n_1
Cy=0	CALL NC, n_1n_2	D4, n_2, n_1
P/Vフラグの内容により		
P/V=1	CALL PE, n_1n_2	EC, n_2, n_1
P/V=0	CALL PO, n_1n_2	E4, n_2, n_1
Sフラグの内容により		
S=1	CALL M, n_1n_2	FC, n_2, n_1
S=0	CALL P, n_1n_2	F4, n_2, n_1



フラグの内容によりメインルーチンへ復帰するRET命令

		インストラクション・コード
Zフラグの内容により		
Z=1	RET Z	C8
Z=0	RET NZ	C0
Cyフラグの内容により		
Cy=1	RET C	D8
Cy=0	RET NC	D0
P/Vフラグの内容により		
P/V=1	RET PE	E8
P/V=0	RET PO	E0
Sフラグの内容により		
S=1	RET M	F8
S=0	RET P	F0

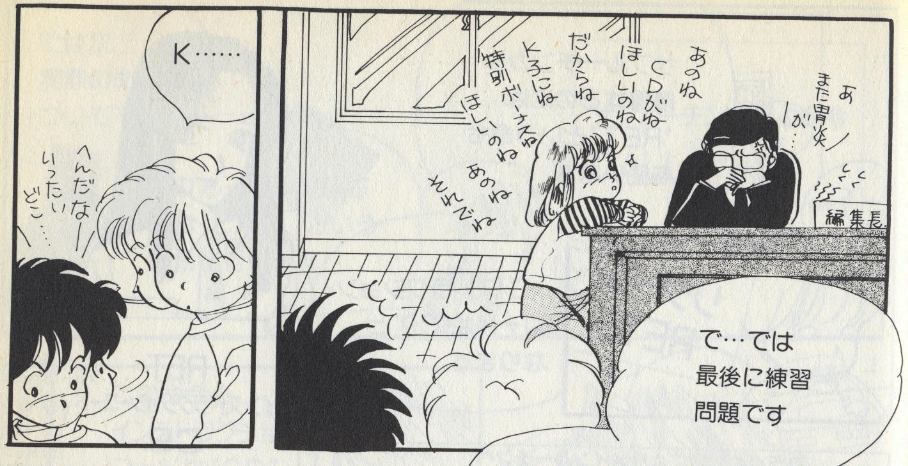
問題　Bレジスタの内容をカウンタにして，初期値より2倍になったらメインルーチンへ復帰するサブプログラムを作成せよ。(一種のディレイルーチンのようなもの)

答 (例1)

```
SUB     LD A,B
        ADD A,A
SUB1    INC B
        CP B
        RET Z
        JR SUB1
```

(例2)

```
SUB     LD A,B
        ADD A,A
SUB1    INC B
        CP B
        JR NZ,SUB1
        RET
```



（問１） 次のプログラムの流れを番号で答えよ。
（例） ①→⑤→④→③→②

メインプログラム

ルーチン１	①
ルーチン２	②
CALL サブ１	③
ルーチン３	④
ルーチン４	⑤

サブプログラム

ルーチン５	⑥
ルーチン６	⑦
CALL サブ２	⑧
RET	⑨

サブ１

ルーチン７	⑩
ルーチン８	⑪
RET	⑫

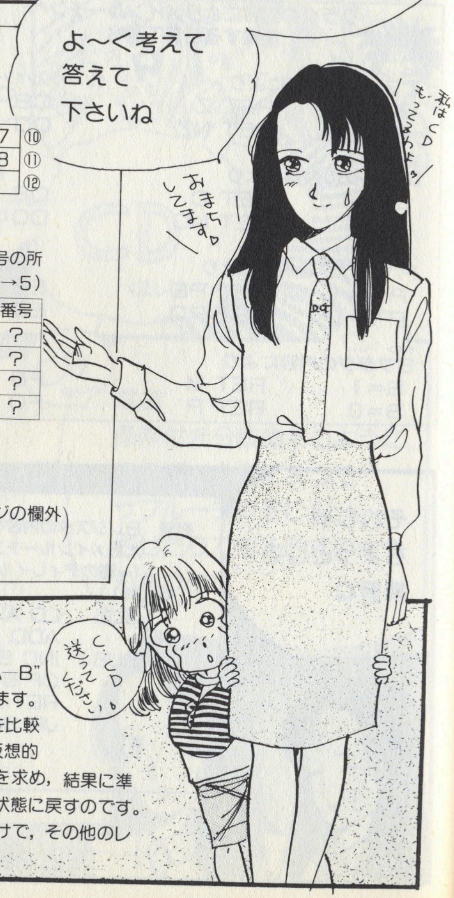
サブ２

（問２） 次のプログラムを実行するとnの値によりどの番号の所へ分岐するでしょうか。その番号で答えよ。（例１→５）

```
LD A,n          SUB ADD A,A
CALL SUB            LD  B,A
CP 10               ADD A,A
JR Z,①              ADD A,A
CP 20               ADD A,B
JR Z,②              RET
CP 30
JR Z,③
CP 100
JR C,④
JR ⑤
```

n	番号
1	?
3	?
5	?
10	?

（答えは108ページの欄外です。）

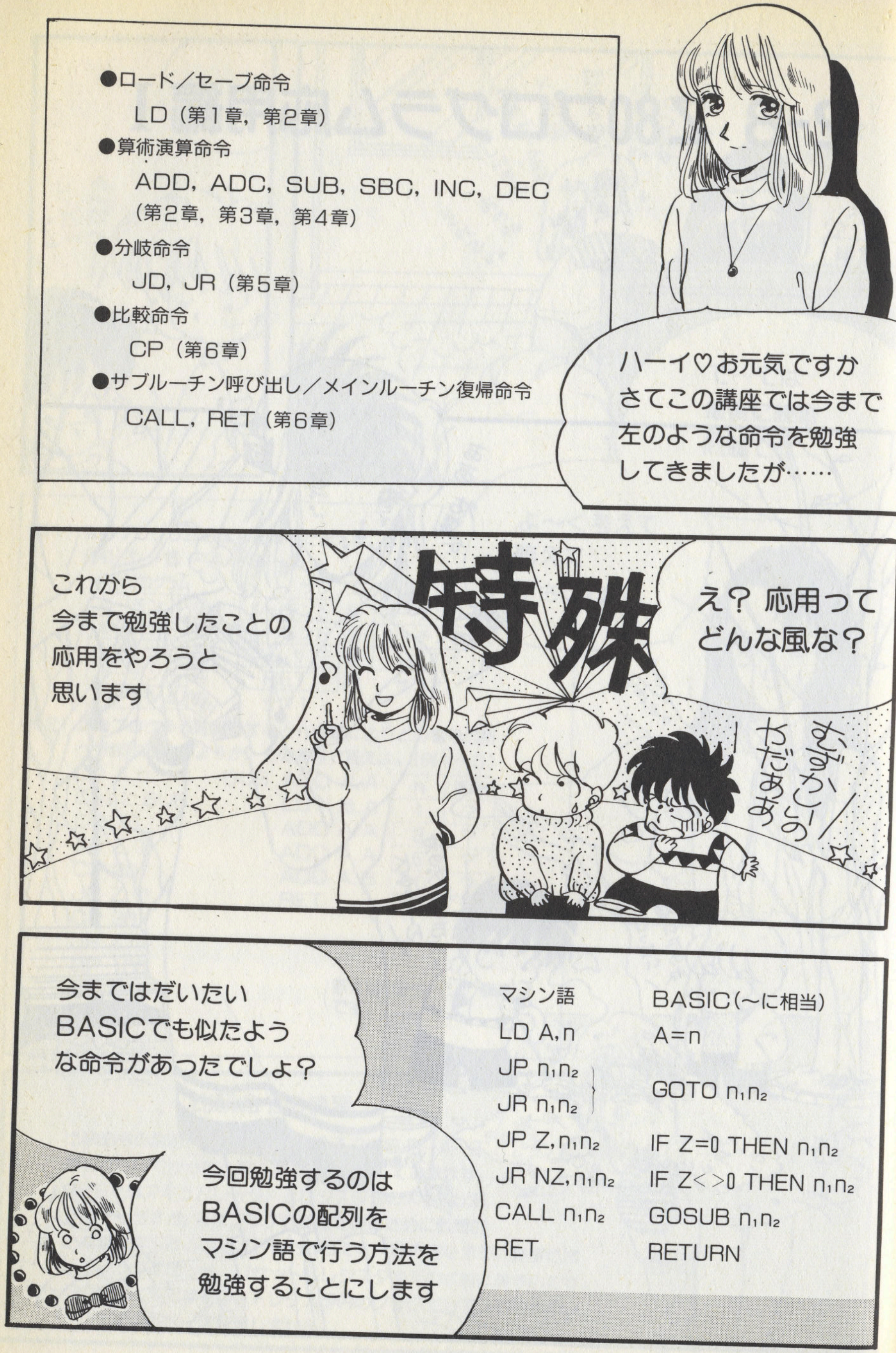


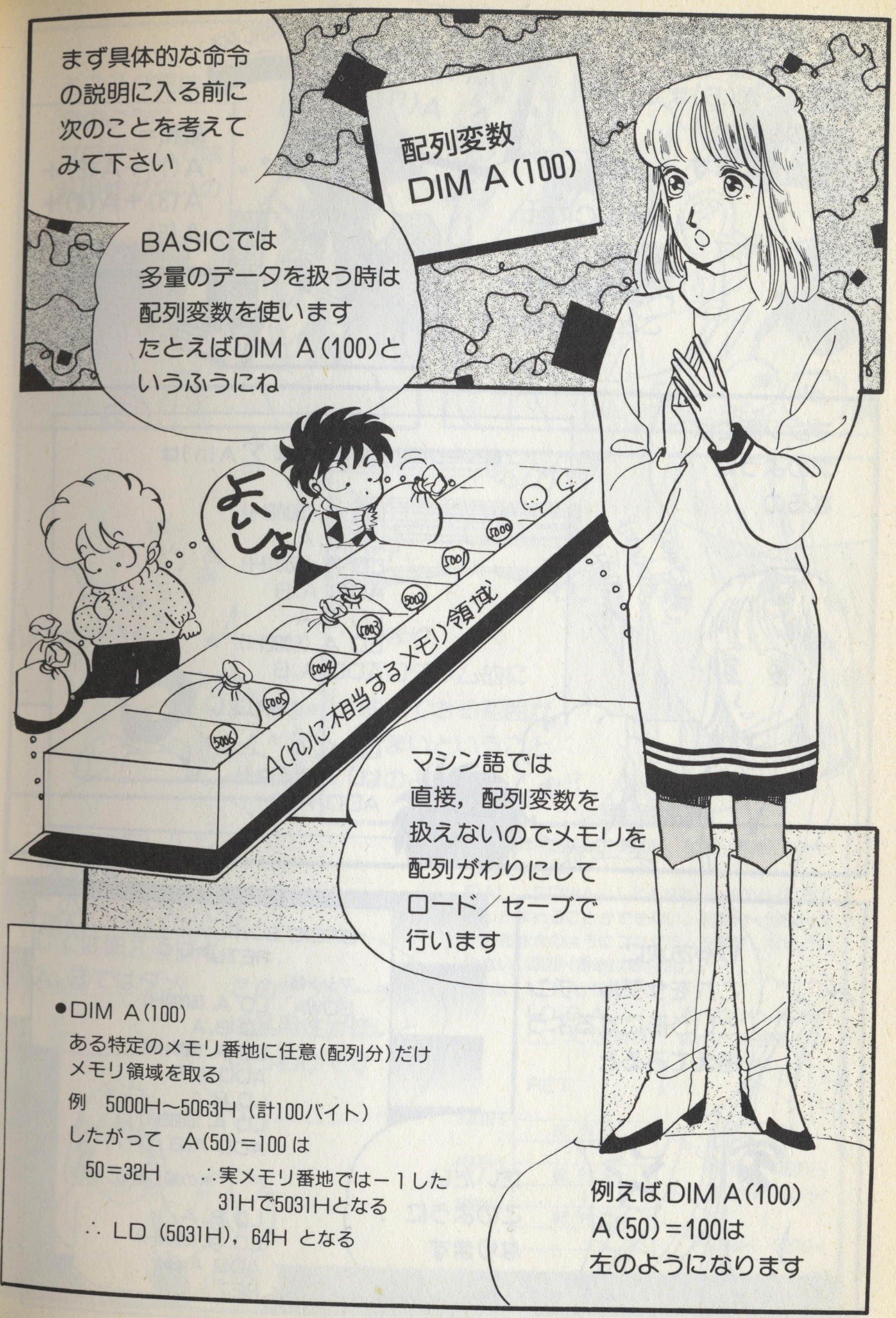
比較命令の補習

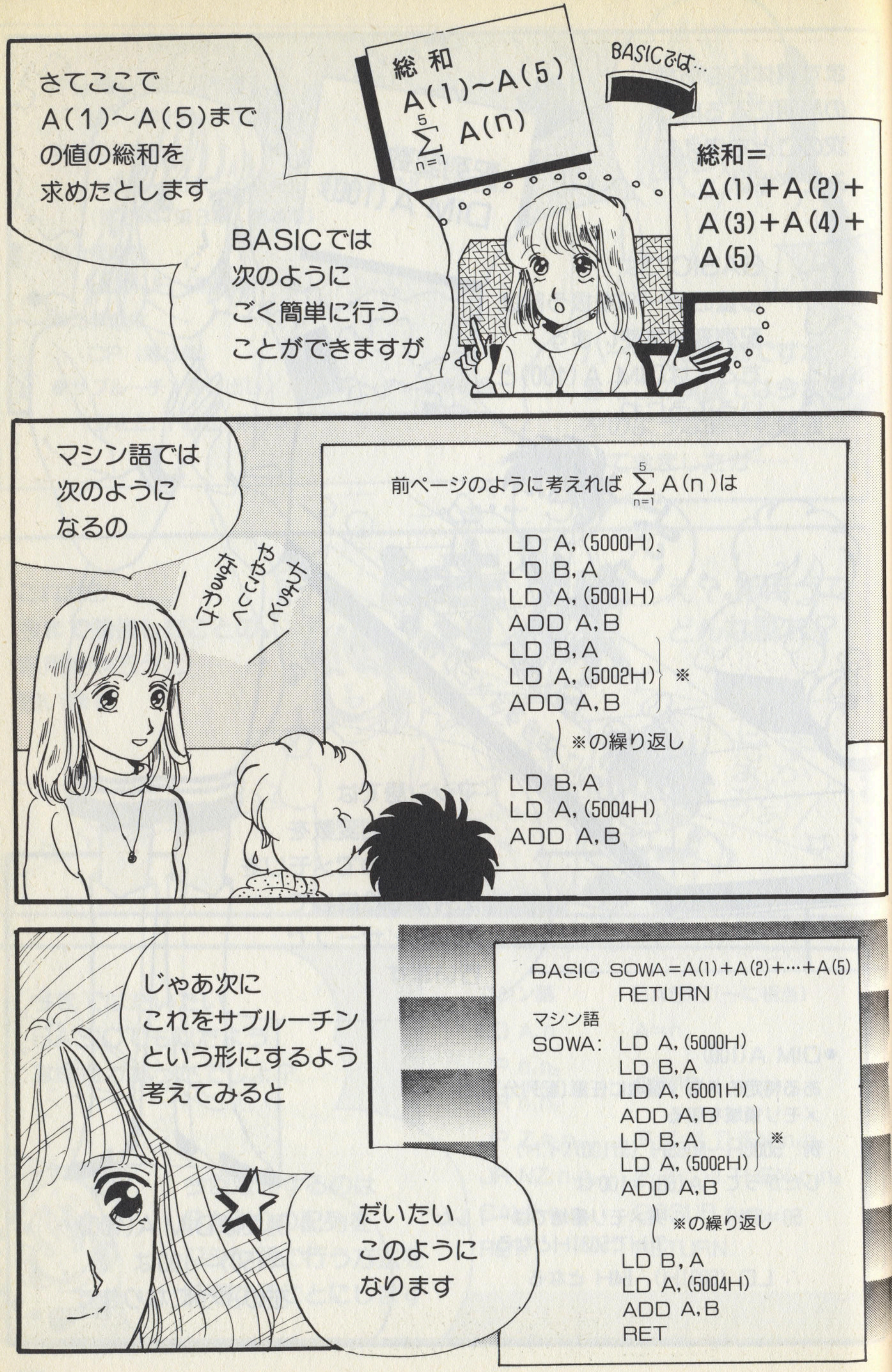
比較命令の説明（100ページ）のところで"CP B→A—B"がわかりにくかったと思いますので補足しておきます。CPBとはアキュムレータAとレジスタBの大小を比較する命令ですが，各種フラグの変化を得るために仮想的に減算を実行するのです。したがってA－Bの差を求め，結果に準じてフラグをセット／リセットしレジスタを元の状態に戻すのです。その結果，変化するのはFレジスタ内のフラグだけで，その他のレジスタの内容は一切変化しません。

2-8 Z80プログラム応用編 I

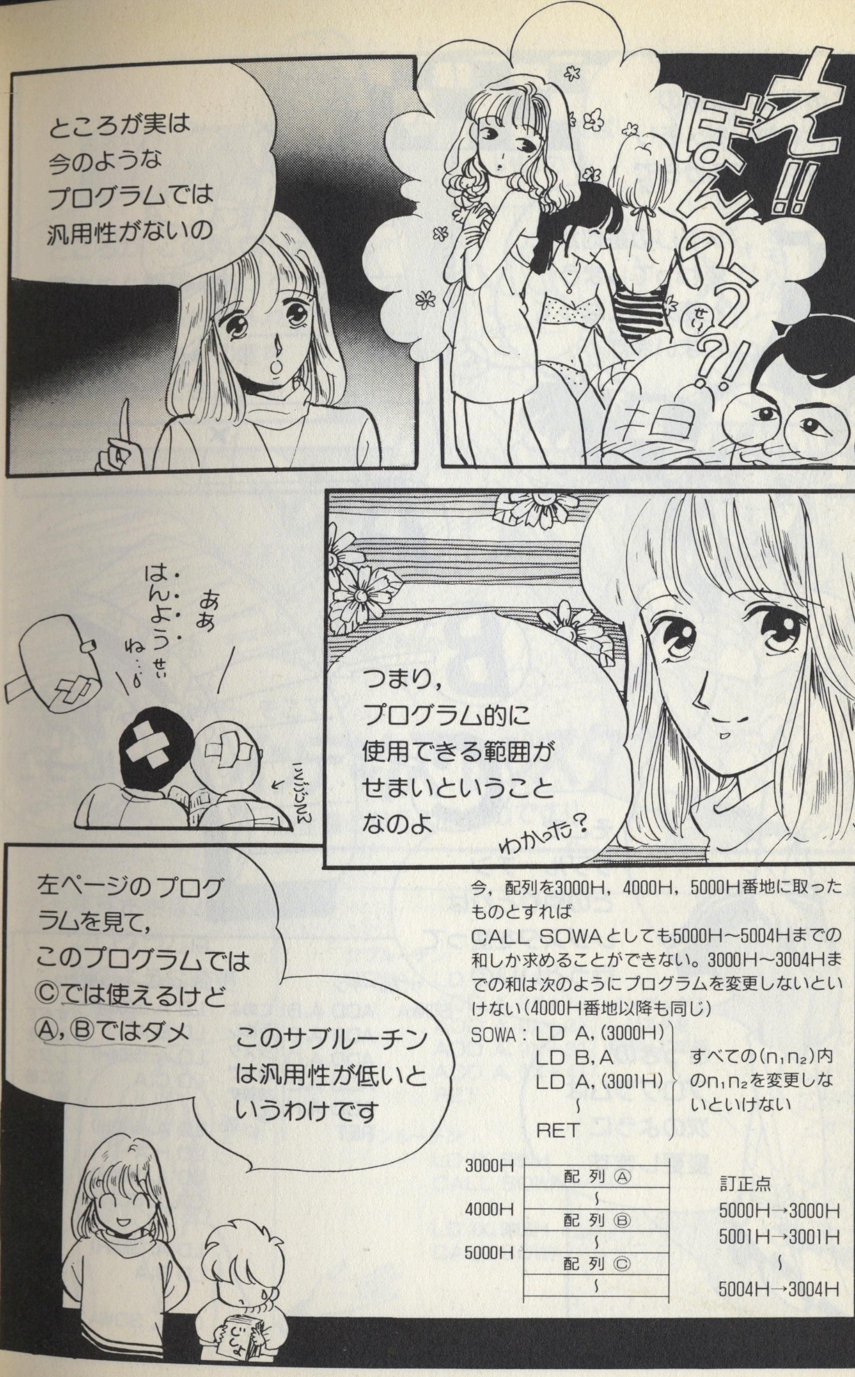



ハーイ♡お元気ですか
さてこの講座では今まで
左のような命令を勉強
してきましたが……
●ロード／セーブ命令
LD（第1章，第2章）
●算術演算命令
ADD，ADC，SUB，SBC，INC，DEC
（第2章，第3章，第4章）
●分岐命令
JD，JR（第5章）
●比較命令
CP（第6章）
●サブルーチン呼び出し／メインルーチン復帰命令
CALL，RET（第6章）
これから
今まで勉強したことの
応用をやろうと
思います
特殊
え？ 応用って
どんな風な？
むずかしーの
やだあああ
今まではだいたい
BASICでも似たよう
な命令があったでしょ？
マシン語 BASIC（〜に相当）
LD A,n A=n
JP n1n2 GOTO n1n2
JR n1n2
JP Z,n1n2 IF Z=0 THEN n1n2
JR NZ,n1n2 IF Z<>0 THEN n1n2
CALL n1n2 GOSUB n1n2
RET RETURN
今回勉強するのは
BASICの配列を
マシン語で行う方法を
勉強することにします

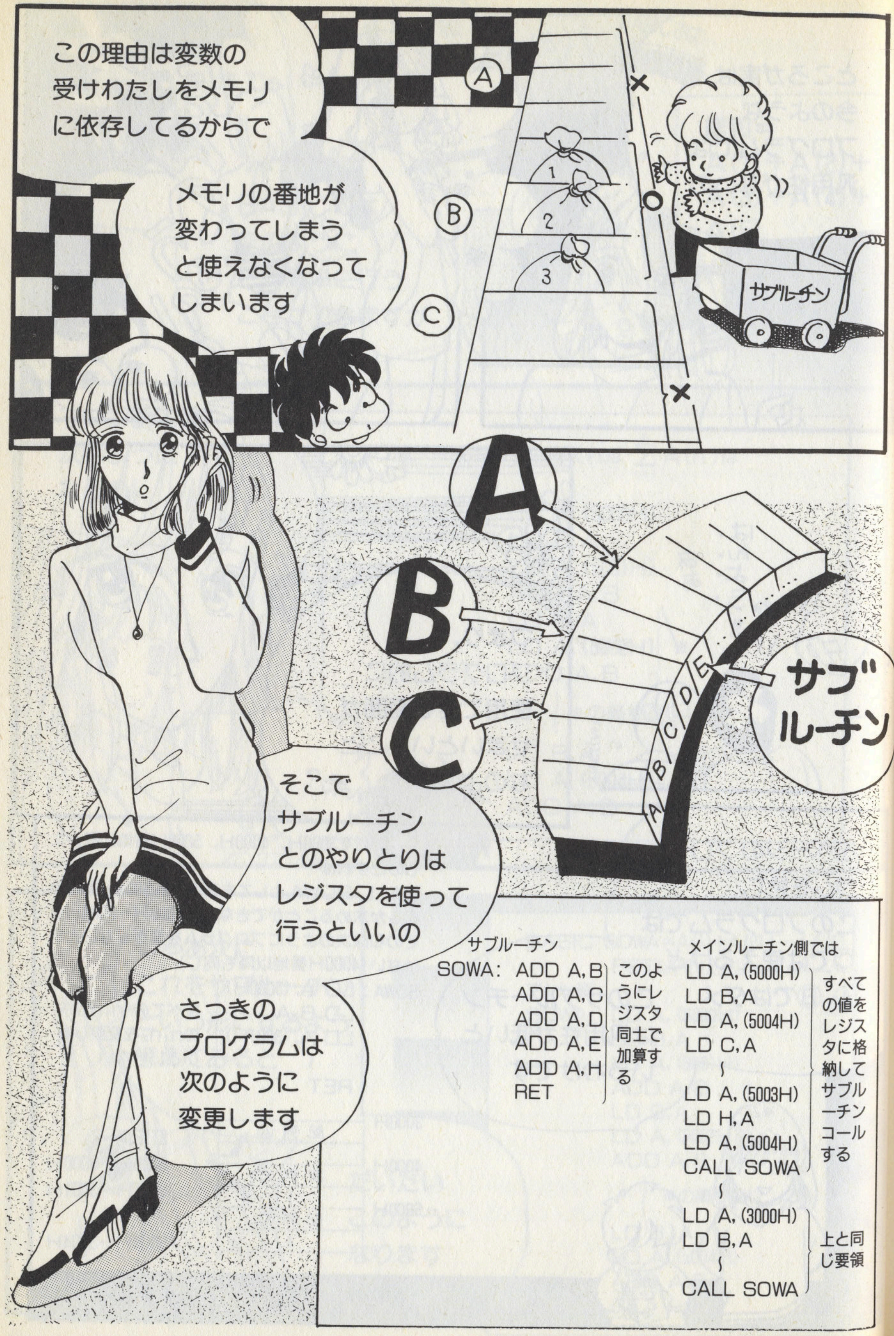

まず具体的な命令
の説明に入る前に
次のことを考えて
みて下さい
配列変数
DIM A(100)
BASICでは
多量のデータを扱う時は
配列変数を使います
たとえばDIM A(100)と
いうふうにね
よいしょ
A(n)に相当するメモリ領域
マシン語では
直接，配列変数を
扱えないのでメモリを
配列がわりにして
ロード／セーブで
行います
●DIM A(100)
ある特定のメモリ番地に任意（配列分）だけ
メモリ領域を取る
例 5000H〜5063H（計100バイト）
したがって A(50)=100 は
50=32H ∴実メモリ番地では−1した
31Hで5031Hとなる
∴ LD (5031H)，64H となる
例えばDIM A(100)
A(50)=100は
左のようになります



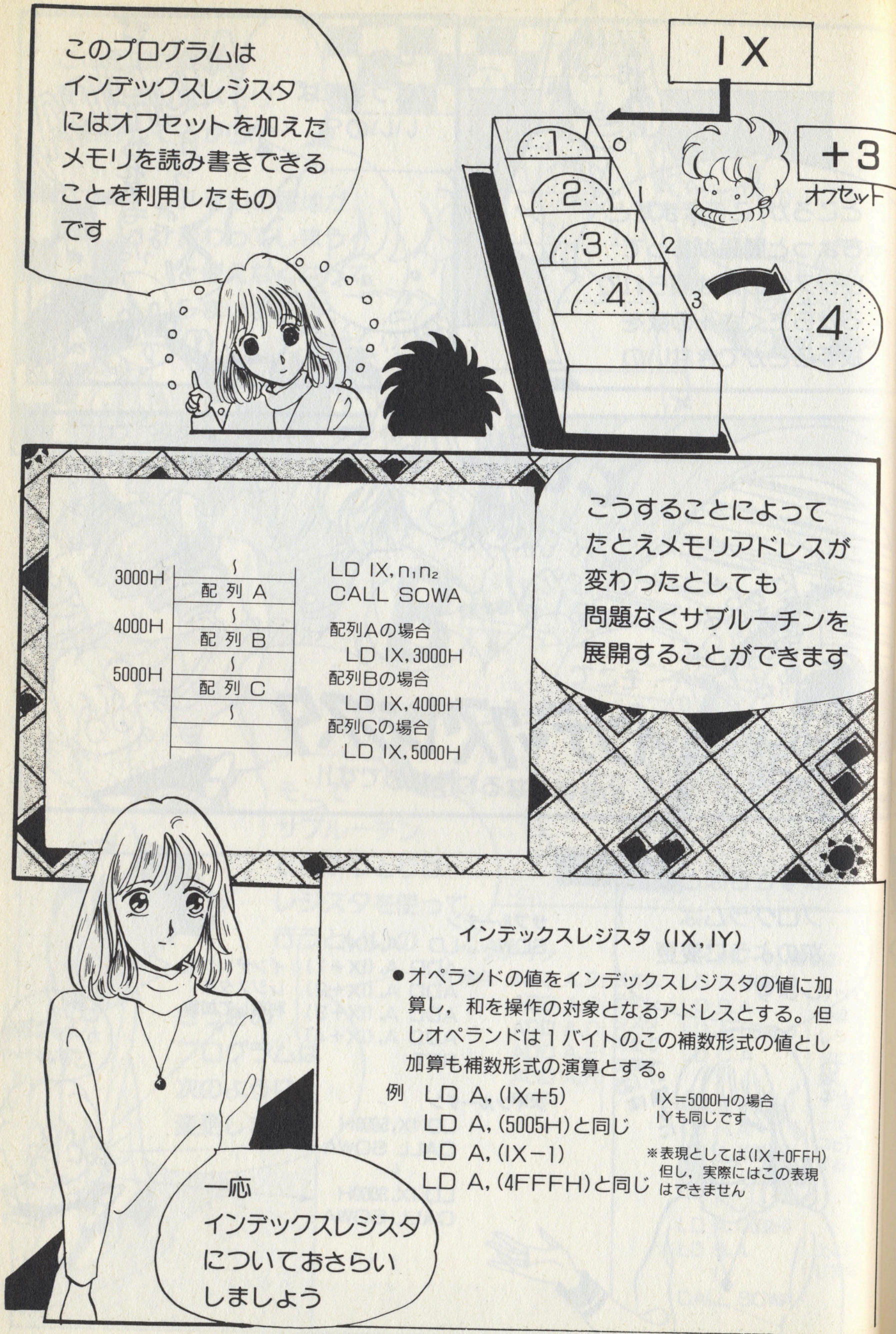
さてここで
A(1)~A(5)まで
の値の総和を
求めたとします
総和
A(1)~A(5)
$\sum_{n=1}^{5}$ A(n)
BASICでは…
総和=
A(1)+A(2)+
A(3)+A(4)+
A(5)
BASICでは
次のように
ごく簡単に行う
ことができますが
マシン語では
次のように
なるの
ちょっとややこしくなるわけ
前ページのように考えれば $\sum_{n=1}^{5}$ A(n)は
LD A,(5000H)
LD B,A
LD A,(5001H)
ADD A,B
LD B,A
LD A,(5002H)
ADD A,B
※
※の繰り返し
LD B,A
LD A,(5004H)
ADD A,B
じゃあ次に
これをサブルーチン
という形にするよう
考えてみると
BASIC SOWA=A(1)+A(2)+…+A(5)
RETURN
マシン語
SOWA: LD A,(5000H)
LD B,A
LD A,(5001H)
ADD A,B
LD B,A
LD A,(5002H)
ADD A,B
※
※の繰り返し
LD B,A
LD A,(5004H)
ADD A,B
RET
だいたい
このように
なります



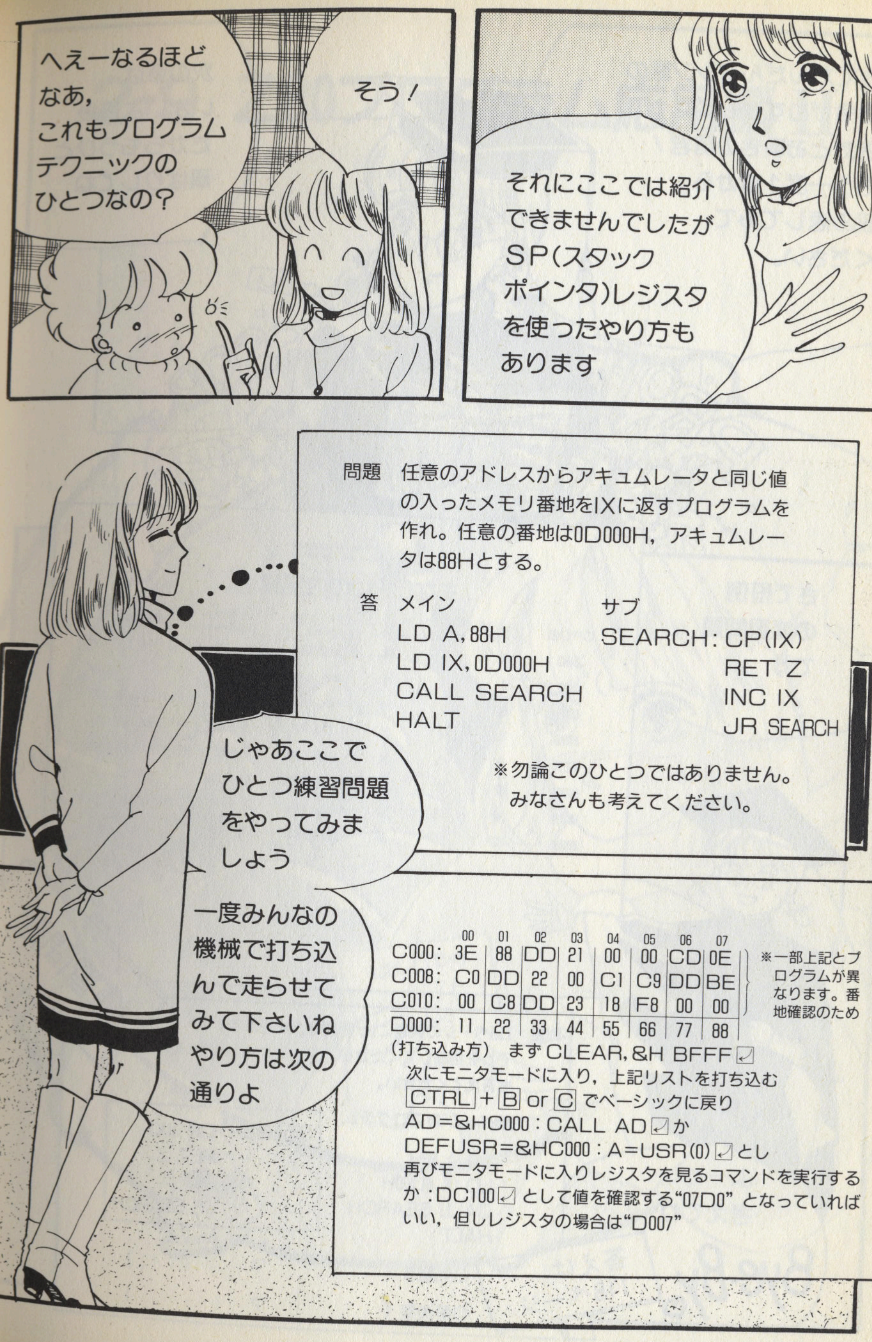
ところが実は
今のような
プログラムでは
汎用性がないの
え!!
ほんのう??!!
せい
ああ
はん・よう・せい・・・ね・・・
つまり,
プログラム的に
使用できる範囲が
せまいということ
なのよ
わかった?
左ページのプログ
ラムを見て,
このプログラムでは
Ⓒでは使えるけど
Ⓐ,Ⓑではダメ
このサブルーチン
は汎用性が低いと
いうわけです
今,配列を3000H,4000H,5000H番地に取った
ものとすれば
CALL SOWAとしても5000H~5004Hまでの
和しか求めることができない。3000H~3004Hま
での和は次のようにプログラムを変更しないとい
けない(4000H番地以降も同じ)
SOWA: LD A,(3000H)
LD B,A
LD A,(3001H)
RET
すべての(n_1n_2)内
のn_1n_2を変更しな
いといけない
3000H 配列Ⓐ
4000H 配列Ⓑ
5000H 配列Ⓒ
訂正点
5000H→3000H
5001H→3001H
5004H→3004H


この理由は変数の
受けわたしをメモリ
に依存してるからで
メモリの番地が
変わってしまう
と使えなくなって
しまいます
A
B
C
1
2
3
サブルーチン
A
B
C
A B C D E
サブ
ルーチン
そこで
サブルーチン
とのやりとりは
レジスタを使って
行うといいの
さっきの
プログラムは
次のように
変更します
サブルーチン
SOWA: ADD A, B
ADD A, C
ADD A, D
ADD A, E
ADD A, H
RET
このようにレジスタ同士で加算する
メインルーチン側では
LD A, (5000H)
LD B, A
LD A, (5004H)
LD C, A
∫
LD A, (5003H)
LD H, A
LD A, (5004H)
CALL SOWA
∫
LD A, (3000H)
LD B, A
∫
CALL SOWA
すべての値をレジスタに格納してサブルーチンコールする
上と同じ要領

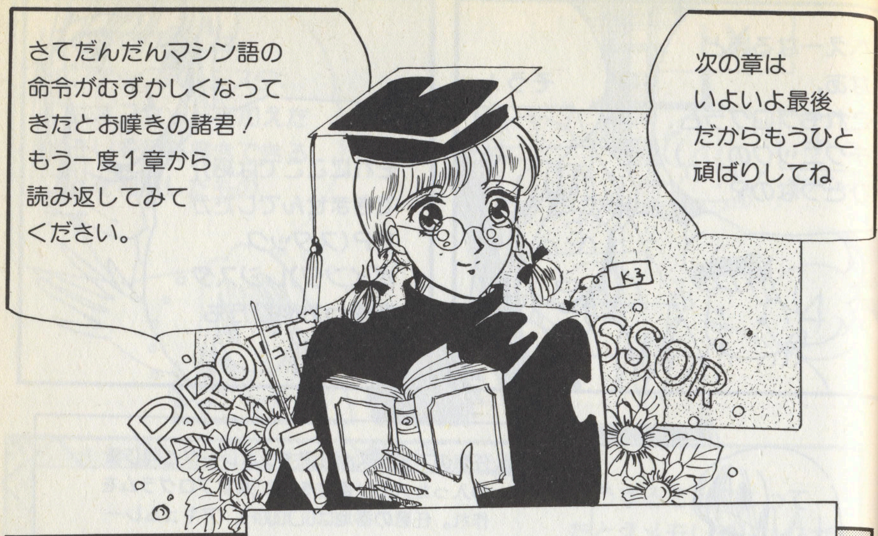

ところが，このままだと
ちょっと問題があって
レジスタの数は限られて
るからたくさんの数を
扱うことができないの
じゃ
どうすれば
いいの？
MURIDA!!
やれるもんなら
やってみなっ
ハイッ!
どど————ん
海ぶすき——っ
お〜 ついにでましたついに K子の切り札!!
そこで
インデックスレジスタ
の登場となるわけなのです!!
まずさきほどの
プログラムは
次のように変更
します
はは 調子に
のって
しまった
サブルーチン
SOWA: LD A, (IX)
ADD A, (IX+1)
ADD A, (IX+2)
ADD A, (IX+3)
ADD A, (IX+4)
RET
インデックス
レジスタを
利用して加算
メインルーチン
LD IX, 5000H
CALL SOWA
∫
LD IX, 3000H
CALL SOWA
ここが
かわった部分
だびょ〜ん

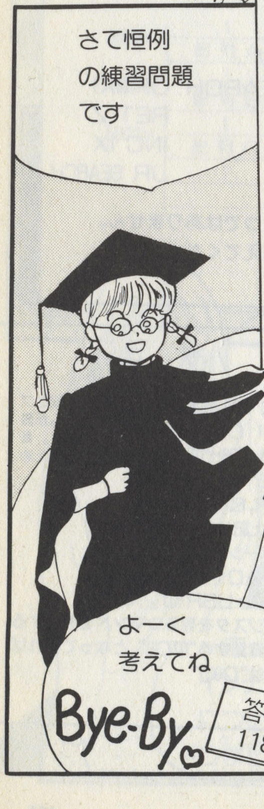


このプログラムはインデックスレジスタにはオフセットを加えたメモリを読み書きできることを利用したものです
IX
+3
オフセット
1
2
3
4
0
1
2
3
4
こうすることによってたとえメモリアドレスが変わったとしても問題なくサブルーチンを展開することができます
3000H
配列A
4000H
配列B
5000H
配列C
LD IX, n_1n_2
CALL SOWA
配列Aの場合
LD IX,3000H
配列Bの場合
LD IX,4000H
配列Cの場合
LD IX,5000H
インデックスレジスタ (IX,IY)
●オペランドの値をインデックスレジスタの値に加算し，和を操作の対象となるアドレスとする。但しオペランドは1バイトのこの補数形式の値とし加算も補数形式の演算とする。
例 LD A,(IX+5)
LD A,(5005H)と同じ
LD A,(IX−1)
LD A,(4FFFH)と同じ
IX=5000Hの場合
IYも同じです
※表現としては(IX+0FFH)
但し，実際にはこの表現はできません
一応インデックスレジスタについておさらいしましょう



へえーなるほどなあ，これもプログラムテクニックのひとつなの？
そう！
それにここでは紹介できませんでしたがSP(スタックポインタ)レジスタを使ったやり方もあります
問題　任意のアドレスからアキュムレータと同じ値の入ったメモリ番地をIXに返すプログラムを作れ。任意の番地は0D000H，アキュムレータは88Hとする。
答　メイン
LD A,88H
LD IX,0D000H
CALL SEARCH
HALT
サブ
SEARCH: CP(IX)
RET Z
INC IX
JR SEARCH
※勿論このひとつではありません。みなさんも考えてください。
じゃあここでひとつ練習問題をやってみましょう
一度みんなの機械で打ち込んで走らせてみて下さいねやり方は次の通りよ
00 01 02 03 04 05 06 07
C000: 3E 88 DD 21 00 00 CD 0E
C008: C0 DD 22 00 C1 C9 DD BE
C010: 00 C8 DD 23 18 F8 00 00
D000: 11 22 33 44 55 66 77 88
※一部上記とプログラムが異なります。番地確認のため
(打ち込み方)　まずCLEAR,&H BFFF⏎
次にモニタモードに入り，上記リストを打ち込む
CTRL + B or C でベーシックに戻り
AD=&HC000：CALL AD⏎か
DEFUSR=&HC000：A=USR(0)⏎とし
再びモニタモードに入りレジスタを見るコマンドを実行するか：DC100⏎として値を確認する“07D0”となっていればいい，但しレジスタの場合は“D007”





主なCPUの内部構造

CPU名	データレジスタのビット数	データバスのビット数	アドレスバスのビット数	アドレス空間 ハード的/ソフト的	クロック（MHz）	代表機種
Z80	8	8	16	64K/64K	4〜8	PC88シリーズ，X1シリーズ
6502	8	8	16	64K/64K	1〜2	アップル，ファミコン
6809	8	8	16	64K/64K	1〜2	FM77/AV40、20
8086	16	16	20	1M/1M	5〜10	PC-9801（初期）
80286	16	16	24	16M/16M	8〜12	PC-9801VX，FMR-60
68000	32	16	24	16M/4096M	8〜12.5	X68000
68020	32	32	32	4096M/4096M	12.5〜25	Machintosh II
80386	32	32	32	4096M/64T（ギガ）	12〜20	PC-98XL²，FMR-70 IBM55モデル5570

練習問題

問題　115ページの例題に1ヶ所問題があります。それを指摘してください。又，どうすれば良いかも答えてください。

前ページのプログラム

```
LD A,88H
LD IX,0D00H
CALL SEARCH
HALT
```

?

```
SEARCH: CP (IX)
        RET Z
        INC IX
        JR SEARCH
```

○問題点（　　　　?　　　　）

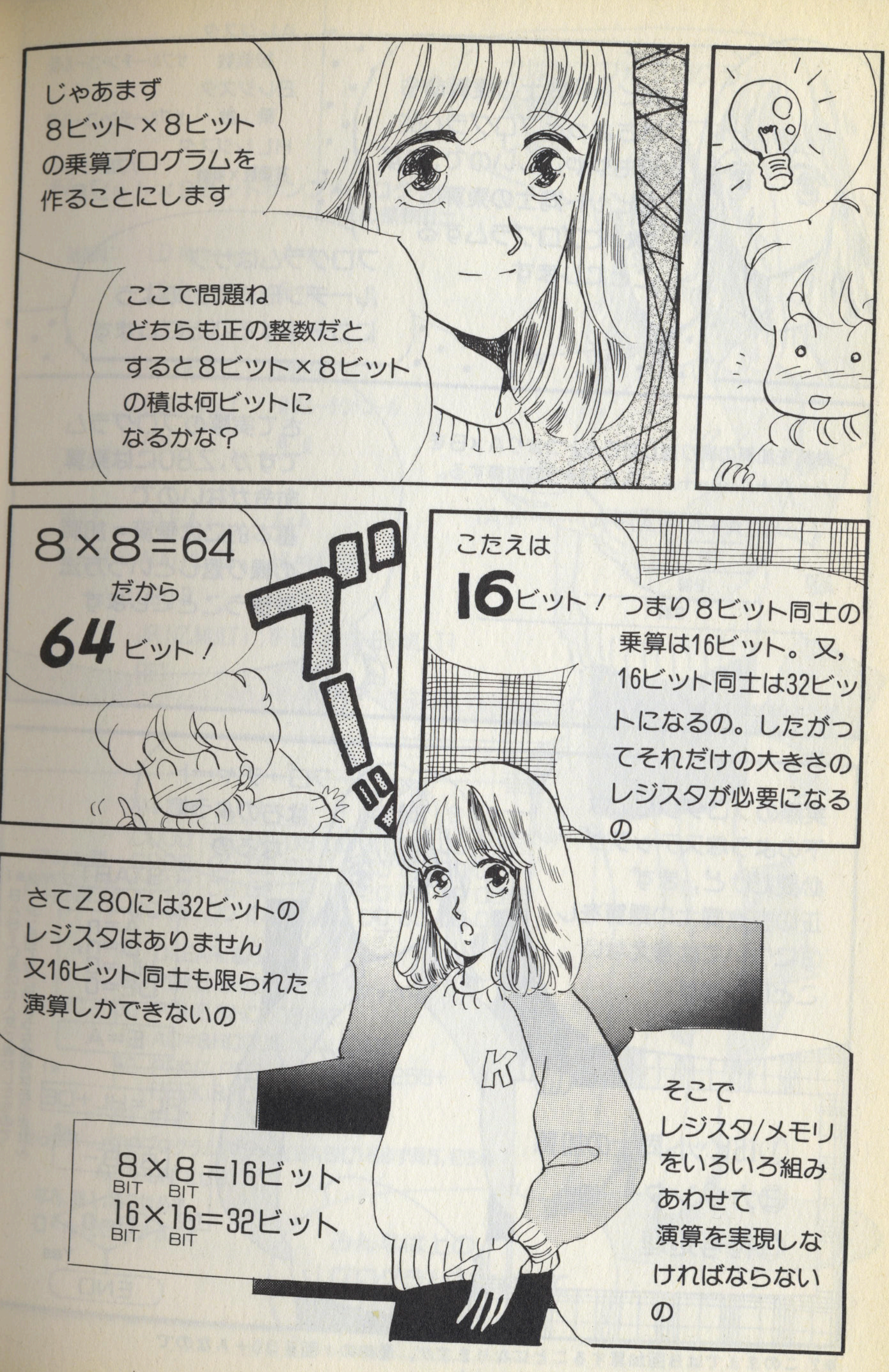
○解決策（　　　　?　　　　）

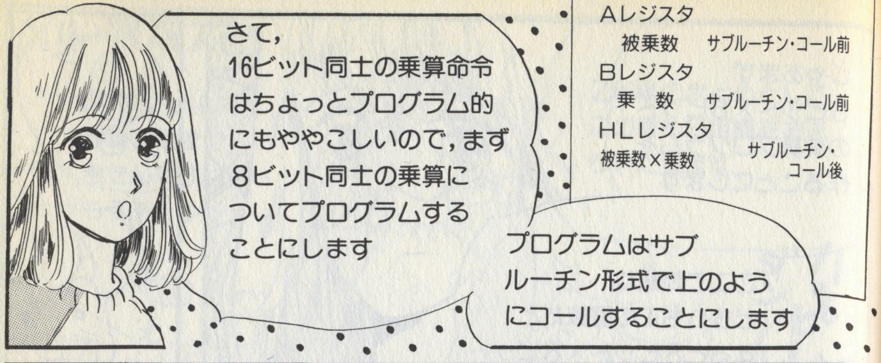
2-9 Z80プログラム応用編 II





●116ページの答え● ①目的の値を発見できなければ永久にプログラムが回り続けてしまう。
②適当なレジスタを使ってカウンタにして停止させるか、サーチする上限を決める。

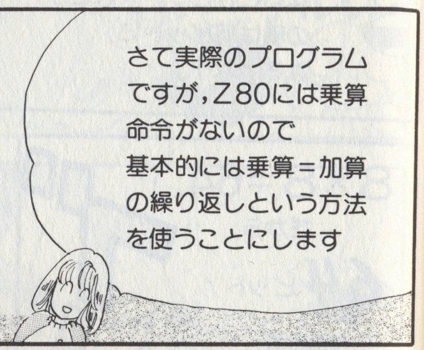


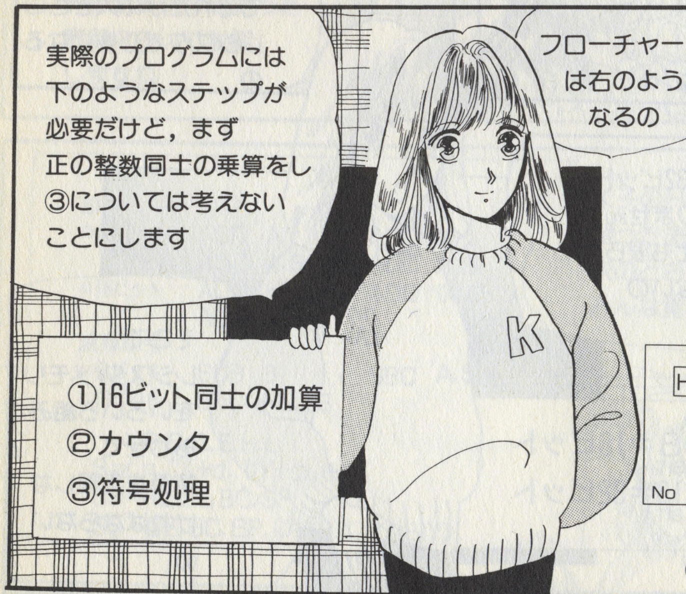


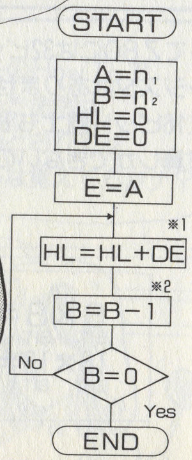
Aレジスタ
　被乗数　サブルーチン・コール前
Bレジスタ
　乗　数　サブルーチン・コール前
HLレジスタ
　被乗数×乗数　サブルーチン・コール後

乗算を加算の繰り返しで行う。つまりA×Bを
A+A+A……+AとAをB−1回加算する。

(……(((A+A)+A)+A)………+A)
1回
2回
3回
B−1

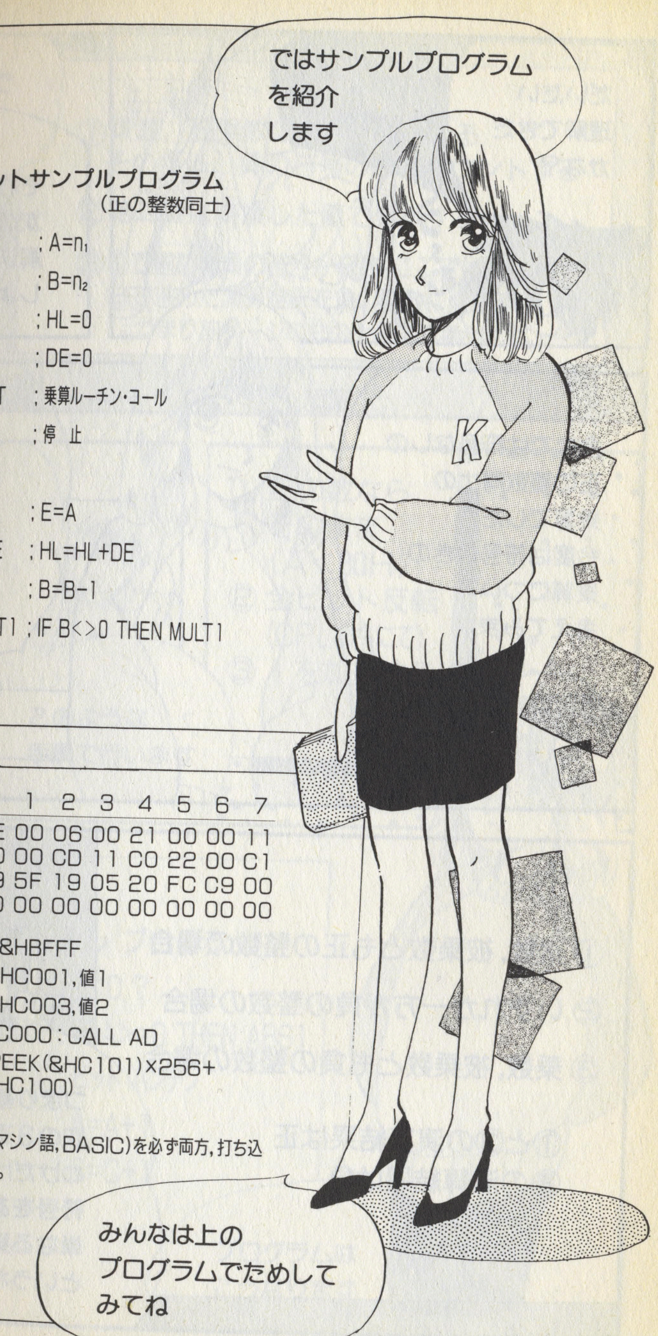






※1 BASIC風の代入文で表わしたもので、マシン語ではこのような命令はありません

※2 このままではB回加算することになりますが，最初の1回目は0+Aなので結局B−1回加算するのと同じことです。



8ビット×8ビットサンプルプログラム
（正の整数同士）

```
MAIN:   LD A,n1        ; A=n1
        LD B,n2        ; B=n2
        LD HL,0        ; HL=0
        LD DE,0        ; DE=0
        CALL MULT      ; 乗算ルーチン・コール
        HALT           ; 停 止

MULT:   LD E,A         ; E=A
MULT1:  ADD HL,DE      ; HL=HL+DE
        DEC B          ; B=B−1
        JR NZ,MULT1    ; IF B<>0 THEN MULT1
        RET
```

マシン語（必ずC000Hから打ち込むこと）

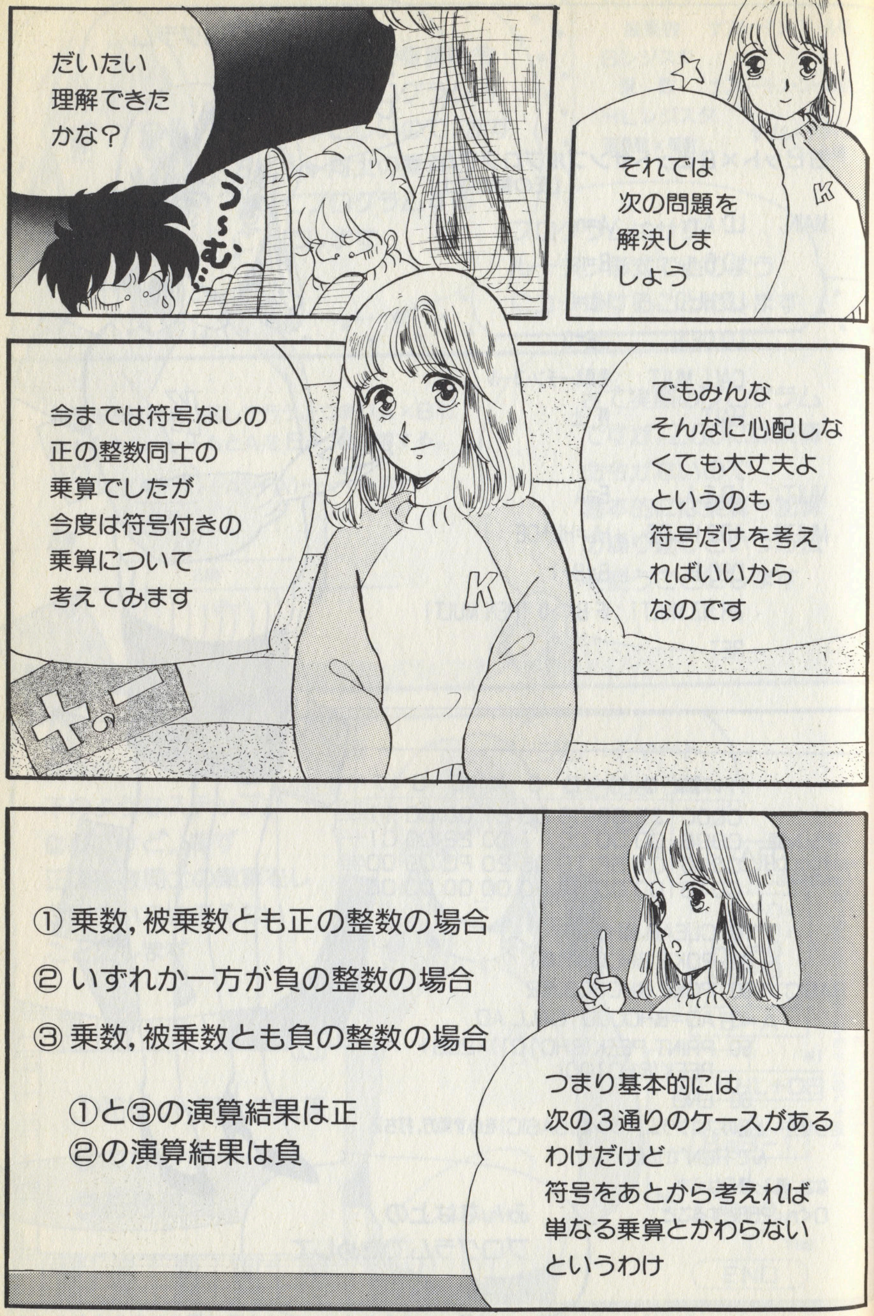
アドレス番地	0	1	2	3	4	5	6	7
C000	3E	00	06	00	21	00	00	11
C008	00	00	CD	11	C0	22	00	C1
C010	C9	5F	19	05	20	FC	C9	00
C018	00	00	00	00	00	00	00	00

BASIC

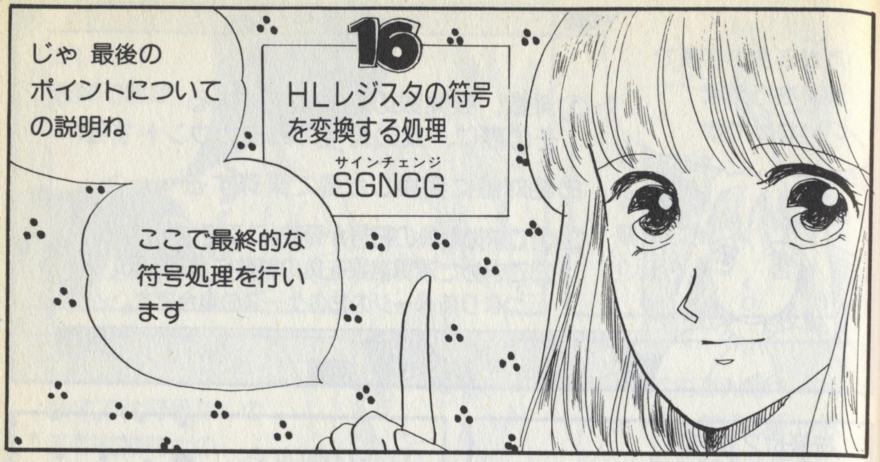
```
10 CLEAR,&HBFFF
20 POKE &HC001,値1
30 POKE &HC003,値2
40 AD=&HC000:CALL AD
50 PRINT PEEK(&HC101)×256+
   PEEK(&HC100)
60 END
```

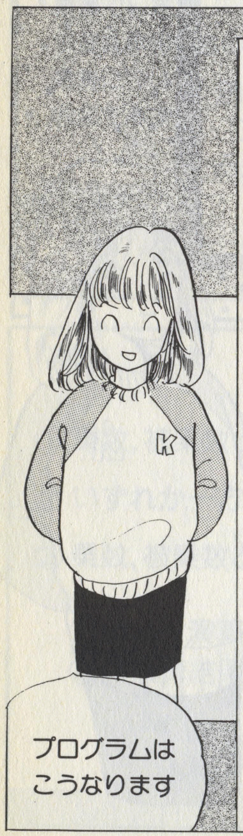
走らせ方：上記のプログラム（マシン語，BASIC）を必ず両方，打ち込んで“RUN”させる。

なお，値1，値2は必ず
0<n<256とすること


だいたい
理解できた
かな？
う～む…
それでは
次の問題を
解決しま
しょう
今までは符号なしの
正の整数同士の
乗算でしたが
今度は符号付きの
乗算について
考えてみます
でもみんな
そんなに心配しな
くても大丈夫よ
というのも
符号だけを考え
ればいいから
なのです
① 乗数，被乗数とも正の整数の場合
② いずれか一方が負の整数の場合
③ 乗数，被乗数とも負の整数の場合
①と③の演算結果は正
②の演算結果は負
つまり基本的には
次の３通りのケースがある
わけだけど
符号をあとから考えれば
単なる乗算とかわらない
というわけ


だから次の手順で
演算を行えば
いいわけです
①乗数，被乗数の絶対値を求める
その際に，負の符号の数をカウントする
②絶対値に変換した値で乗算する
③①で求めた負の符号が奇数(ここでは１)なら
②で求めた演算結果を負の整数に変換する
つまり前ページの②のケースの場合です。
符号付２進数の
符号をとる。
つまり絶対値を
求めるのは
① 正の数なら
そのまま
(A＜80H)
② 全ビット反転
(CPL命令にて)
③ １を加算する
このような
手順で行います
；ゼッタイチ ヲ モトメル プログラム
ABS:　CP 80H　;A<0 ?
JR C,ABS1　;IF A>=0 THEN ABS1
CPL　;ビット ハンテン
INC A　;A=A+1
INC C　;C=C+1
ABS1:　RET
プログラムは
こうなります

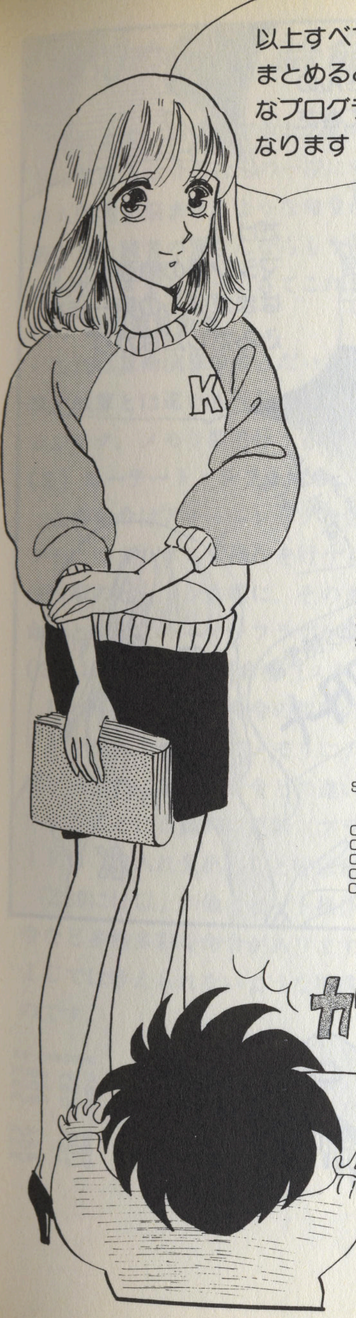





```
; 2バイト ノ HLレジスタ ノ フゴウ ヲ カエル プログラム
; タダシ ジョウザンケッカ ヲ マイナス ニ ヘンカン スル バアイ

SGNCG:   LD A,C          ; A=C
         CP 1            ; C=1 ?
         JR NZ,SGNCG1    ; IF C<>1 THEN SGNCG1
         LD A,H          ; A=H
         CPL             ; ビット ハンテン
         LD H,A          ; H=A
         LD A,L          ; A=L
         CPL             ; ビット ハンテン
         LD L,A          ; L=A
         INC HL          ; HL=HL+1
SGNCG1:  RET
```

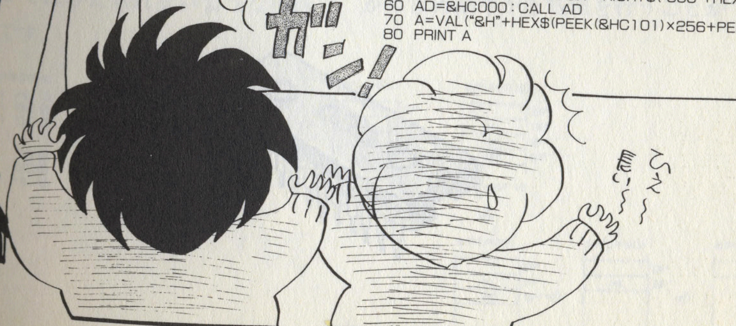
○解説　Cレジスタには負の符号の数が格納されている，もし奇数(ここでは1)なら，HLレジスタを全ビット反転し，＋1を加算して負の数に符号を変換する

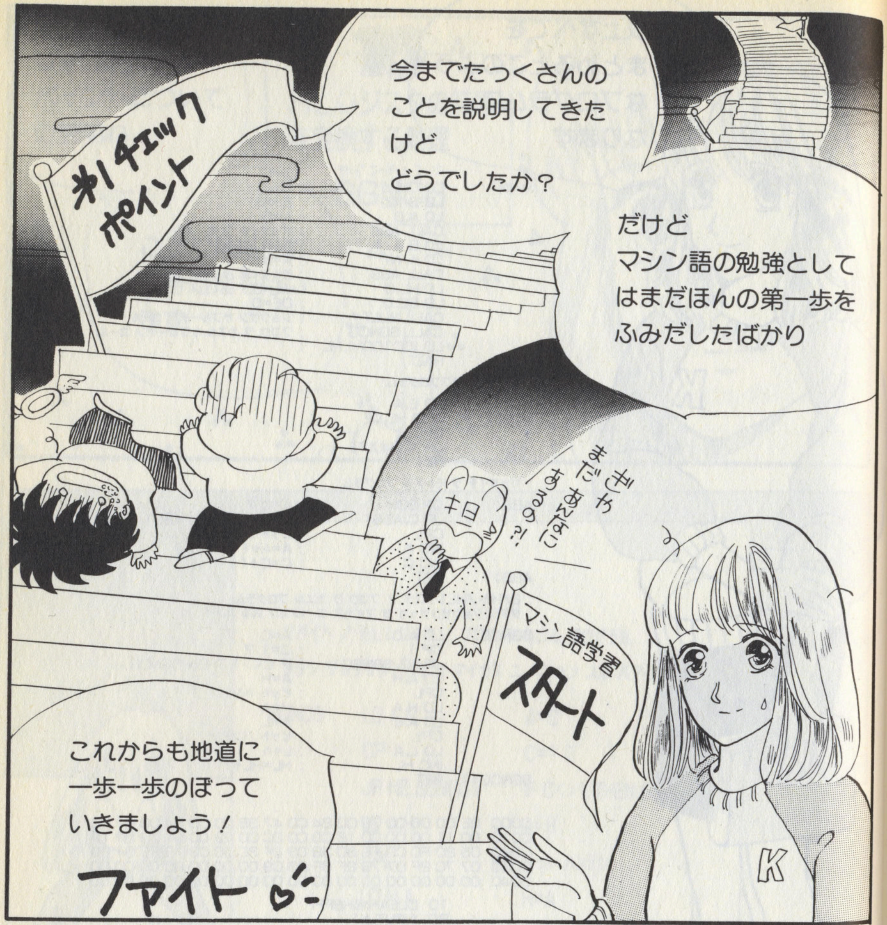


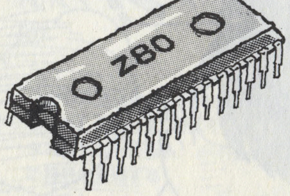

```
MAIN:    LD C,0            ; C=0
         LD B,N2           ; B=N2
         LD A,B            ; A=B
         CALL ABS          ; ゼッタイチ ヲ モトメル
         LD B,A            ; B=A
         LD A,N1           ; A=N1
         CALL ABS          ; ゼッタイチ ヲ モトメル
         LD HL,0           ; HL=0
         LD DE,0           ; DE=0
         CALL MULT         ; ジョウザン サブルーチン コール
         CALL SGNCG        ; フゴウ ヲ カエル サブルーチン コール
         LD (0C100H),HL    ;
         HALT
; ジョウザン サブプログラム
MULT:    LD E,A
MULT1:   ADD HL,DE
         DEC B
         JR NZ,MULT1
         RET
; ゼッタイチ ヲ モトメル プログラム
ABS:     CP 80H            ; A<0 ?
         JR C,ABS1         ; IF A>=0 THEN ABS1
         CPL               ; ビット ハンテン
         INC A             ; A=A+1
         INC C             ; C=C+1
ABS1:    RET
; 2バイトノHLレジスタ ノ フゴウ ヲ カエル プログラム
; タダシ ジョウザンケッカ ヲ マイナス ニ ヘンカン スル バアイ
SGNCG:   LD A,C            ; A=C
         CP 1              ; C=1 ?
         JR NZ,SGNCG1      ; IF C<>1 THEN SGNCG1
         LD A,H            ; A=H
         CPL               ; ビット ハンテン
         LD H,A            ; H=A
         LD A,L            ; A=L
         CPL               ; ビット ハンテン
         LD L,A            ; L=A
         INC HL            ; HL=HL+1
SGNCG1:  RET

C000  0E 00 06 00 78 CD 24 C0 47 3E 00 CD 24 C0 21 00
C010  00 11 00 00 CD 1E C0 CD 2C C0 22 00 C1 C9 5F 19
C020  05 20 FC C9 FE 80 38 03 2F 3C 0C C9 79 FE 01 20
C030  07 7C 2F 67 7D 2F 6F 23 C9 00 00 00 00 00 00 00
C040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

10 CLEAR,&HBFFF
20 INPUT N1
30 INPUT N2
40 POKE &HC003,VAL("&H"+RIGHT$("000"+HEX$(N1),2))
50 POKE &HC00A,VAL("&H"+RIGHT$("000"+HEX$(N2),2))
60 AD=&HC000:CALL AD
70 A=VAL("&H"+HEX$(PEEK(&HC101)×256+PEEK(&HC100)))
80 PRINT A
```







Z80マシン語命令（補足）

これまでに紹介してきたZ80のアセンブリ言語，及びインストラクションコードは数ある命令のほんの一握にすぎません。したがってまだまだ難解でマシン語の入門者を悩ませるような命令が目白押しなのです。しかしここまで問題なく読破された読者の皆さんならもう何も心配させる事はありません。安心して勉強を進めていきましょう。さてこれまでに説明できなかった命令は以下のとおりです。

（1）論理演算命令

これは算術演算命令とだいたい同じ動作をする命令と考えていいのですが，加算／減算とは違いＡＮＤ／ＯＲ／ＸＯＲといった論理演算を，アキュムレータとレジスタ，メモリ間で行うものです。

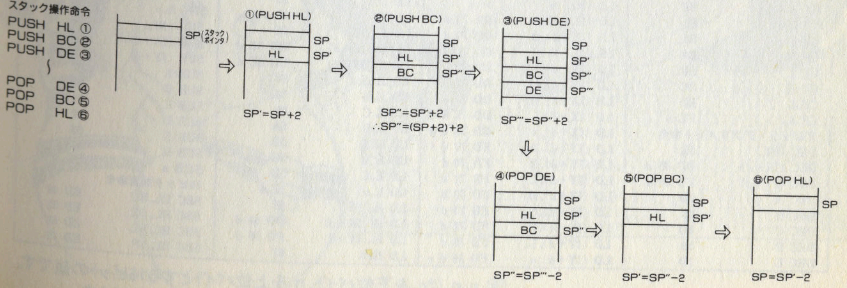
（2）ローテート・シフト命令

この命令はアキュムレータを含む８ビットのレジスタ及び，メモリを左右にシフト，つまりずらす操作を行うものです。ローテート命令の場合は円を描くように左端のビットが右端に，そのまた逆にと回転しながらシフトしていきます。両命令とも間にキャリーフラグが関わっています（ＲＬＤ，ＲＲＤ命令を除く）。

（3）16ビットロード命令（スタック操作に関する命令）

16ビットのロード命令の説明の章では説明しませんでしたが，スタックポインタ・レジスタを介してメモリにデータを格納もしくは取り出すものです。一般の16ビットの“ＬＤ”命令との違いはスタックポインタがデータを格納，取り出しをするごとに自動的に更新（プラス，マイナス）される点にあります。これはＬＩＦＯ（後入れ先出し）と呼ばれるスタックポインタ独特の機能です。

Z80には以上の他にビット操作命令，ブロック転送命令，CPUコントロール命令など多種多彩な命令があります。このようなマシン語を使うことによりＢＡＳＩＣでは考えられないような特殊で，かつきめ細やかな処理を行うことができるのです。

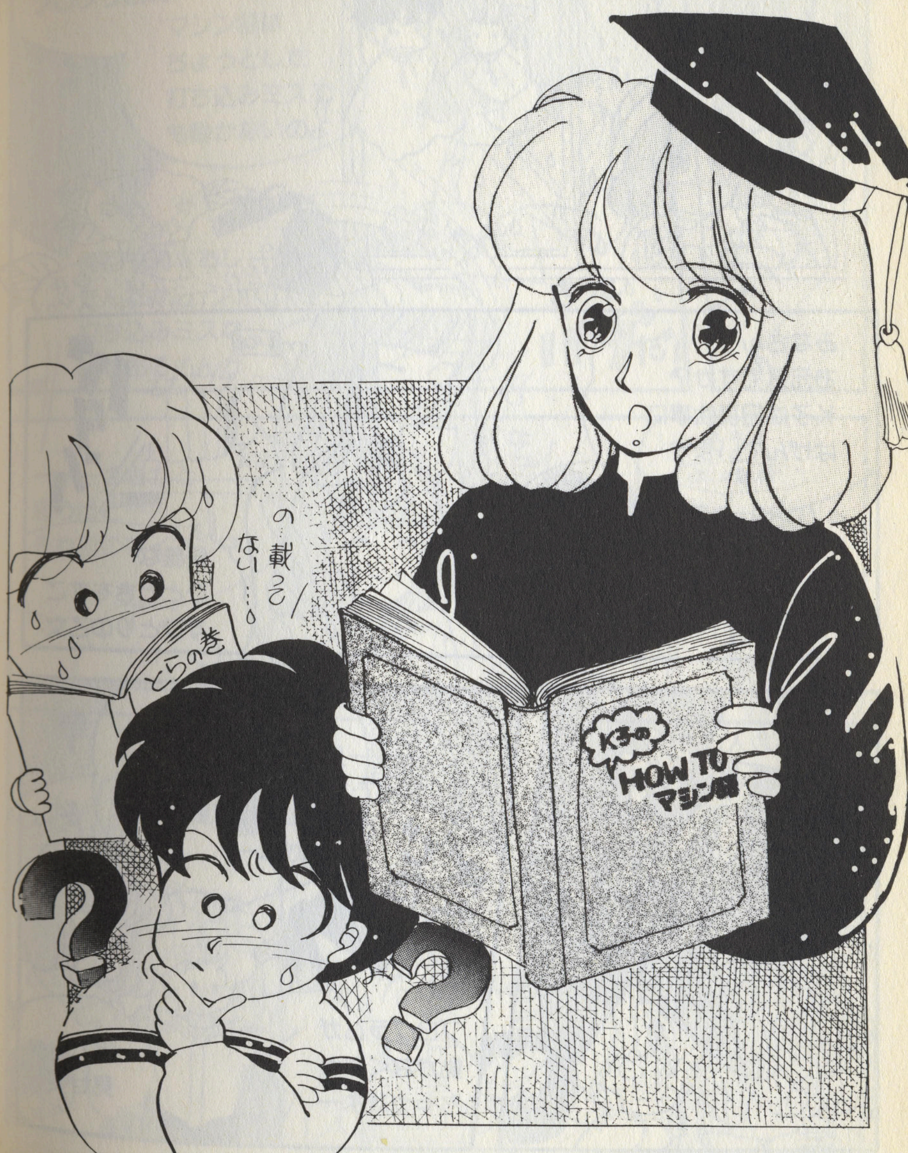


Z80アセンブリ言語←→インストラクションコード対応表

アセンブリ言語←→インストラクションコード	
8ビット加算命令	
ADC A,(HL)	8E
ADC A,(IX+d)	DD 8E d
ADC A,(IY+d)	FD 8E d
ADC A,A	8F
ADC A,B	88
ADC A,C	89
ADC A,D	8A
ADC A,E	8B
ADC A,H	8C
ADC A,L	8D
ADC A,n	CE n
16ビット加算命令	
ADC HL,BC	ED 4A
ADC HL,DE	ED 5A
ADC HL,HL	ED 6A
ADC HL,SP	ED 7A
8ビット加算命令	
ADD A,(HL)	86
ADD A,(IX+d)	DD 86 d
ADD A,(IY+d)	FD 86 d
ADD A,A	87
ADD A,B	80
ADD A,C	81
ADD A,D	82
ADD A,E	83
ADD A,H	84
ADD A,L	85
ADD A,n	C6 n
16ビット加算命令	
ADD HL,BC	09
ADD HL,DE	19
ADD HL,HL	29
ADD HL,SP	39
ADD IX,BC	DD 09
ADD IX,DE	DD 19
ADD IX,IX	DD 29
ADD IX,SP	DD 39
ADD IY,BC	FD 09
ADD IY,DE	FD 19
ADD IY,IY	FD 29
ADD IY,SP	FD 39
サブルーチン呼び出し命令	
CALL C,n_1n_2	DC n_2n_1
CALL M,n_1n_2	FC n_2n_1
CALL NC,n_1n_2	D4 n_2n_1
CALL NZ,n_1n_2	C4 n_2n_1
CALL P,n_1n_2	F4 n_2n_1
CALL PE,n_1n_2	EC n_2n_1
CALL PO,n_1n_2	E4 n_2n_1
CALL Z,n_1n_2	CC n_2n_1
CALL n_1n_2	CD n_2n_1
比較命令	
CP (HL)	BE
CP (IX+d)	DD BE d
CP (IY+d)	FD BE d
CP A	BF
CP B	B8
CP C	B9
CP D	BA
CP E	BB
CP H	BC
CP L	BD
CP n	FE n
8ビット・デクリメント命令	
DEC (HL)	35
DEC (IX+d)	DD 35 d
DEC (IY+d)	FD 35 d
DEC A	3D
DEC B	05
DEC C	0D
DEC D	15
DEC E	1D
DEC H	25
DEC L	2D
16ビット・デクリメント命令	
DEC BC	0B
DEC DE	1B
DEC HL	2B
DEC IX	DD 2B
DEC IY	FD 2B
DEC SP	3B
8ビット・インクリメント命令	
INC (HL)	34
INC (IX+d)	DD 34 d
INC (IY+d)	FD 34 d
INC A	3C
INC B	04
INC C	0C
INC D	14
INC E	1C
INC H	24
INC L	2C
16ビット・インクリメント命令	
INC BC	03
INC DE	13
INC HL	23
INC IX	DD 23
INC IY	FD 23
INC SP	33
ジャンプ命令	
JP (HL)	E9
JP (IX)	DD E9
JP (IY)	FD E9
JP n_1n_2	C3 n_2n_1
JP C,n_1n_2	DA n_2n_1
JP M,n_1n_2	FA n_2n_1
JP NC,n_1n_2	D2 n_2n_1
JP NZ,n_1n_2	C2 n_2n_1
JP P,n_1n_2	F2 n_2n_1
JP PE,n_1n_2	EA n_2n_1
JP PO,n_1n_2	E2 n_2n_1
JP Z,n_1n_2	CA n_2n_1
分岐命令	
JR C,e	38 e
JR NC,e	30 e
JR NZ,e	20 e
JR Z,e	28 e
JR e	18 e
8ビット・ロード命令	
LD (BC),A	02
LD (DE),A	12
LD (HL),A	77
LD (HL),B	70
LD (HL),C	71
LD (HL),D	72
LD (HL),E	73
LD (HL),H	74
LD (HL),L	75
LD (HL),n	36 n
LD (IX+d),A	DD 77 d
LD (IX+d),B	DD 70 d
LD (IX+d),C	DD 71 d
LD (IX+d),D	DD 72 d
LD (IX+d),E	DD 73 d
LD (IX+d),H	DD 74 d
LD (IX+d),L	DD 75 d
LD (IX+d),n	DD 36 d n
LD (IY+d),A	FD 77 d
LD (IY+d),B	FD 70 d
LD (IY+d),C	FD 71 d
LD (IY+d),D	FD 72 d
LD (IY+d),E	FD 73 d
LD (IY+d),H	FD 74 d
LD (IY+d),L	FD 75 d
LD (IY+d),n	FD 36 d n
LD (n_1n_2),A	32 n_2n_1
16ビット・ロード命令	
LD (n_1n_2),BC	ED 43 n_2n_1
LD (n_1n_2),DE	ED 53 n_2n_1
LD (n_1n_2),HL	22 n_2n_1
LD (n_1n_2),IX	DD 22 n_2n_1
LD (n_1n_2),IY	FD 22 n_2n_1
LD (n_1n_2),SP	ED 73 n_2n_1
8ビット・ロード命令	
LD A,(BC)	0A
LD A,(DE)	1A
LD A,(HL)	7E
LD A,(IX+d)	DD 7E d
LD A,(IY+d)	FD 7E d
LD A,(n_1n_2)	3A n_2n_1
LD A,A	7F
LD A,B	78
LD A,C	79
LD A,D	7A
LD A,E	7B
LD A,H	7C
LD A,L	7D
LD A,n	3E n
LD A,I	ED 57
LD A,R	ED 5F
LD B,(HL)	46
LD B,(IX+d)	DD 46 d
LD B,(IY+d)	FD 46 d
LD B,A	47
LD B,B	40
LD B,C	41
LD B,D	42
LD B,E	43
LD B,H	44
LD B,L	45
LD B,n	06 n
LD C,(HL)	4E
LD C,(IX+d)	DD 4E d
LD C,(IY+d)	FD 4E d
LD C,A	4F
LD C,B	48
LD C,C	49
LD C,D	4A
LD C,E	4B
LD C,H	4C
LD C,L	4D
LD C,n	0E n
LD D,(HL)	56
LD D,(IX+d)	DD 56 d
LD D,(IY+d)	FD 56 d
LD D,A	57
LD D,B	50
LD D,C	51
LD D,D	52
LD D,E	53
LD D,H	54
LD D,L	55
LD D,n	16 n
LD E,(HL)	5E
LD E,(IX+d)	DD 5E d
LD E,(IY+d)	FD 5E d
LD E,A	5F
LD E,B	58
LD E,C	59
LD E,D	5A
LD E,E	5B
LD E,H	5C
LD E,L	5D
LD E,n	1E n
LD H,(HL)	66
LD H,(IX+d)	DD 66 d
LD H,(IY+d)	FD 66 d
LD H,A	67
LD H,B	60
LD H,C	61
LD H,D	62
LD H,E	63
LD H,H	64
LD H,L	65
LD H,n	26 n
LD L,(HL)	6E
LD L,(IX+d)	DD 6E d
LD L,(IY+d)	FD 6E d
LD L,A	6F
LD L,B	68
LD L,C	69
LD L,D	6A
LD L,E	6B
LD L,H	6C
LD L,L	6D
LD L,n	2E n
LD I,A	ED 47
LD R,A	ED 4F
16ビット・ロード命令	
LD BC,(n_1n_2)	ED 4B n_2n_1
LD BC,n_1n_2	01 n_2n_1
LD DE,(n_1n_2)	ED 5B n_2n_1
LD DE,n_1n_2	11 n_2n_1
LD HL,(n_1n_2)	2A n_2n_1
LD HL,n_1n_2	21 n_2n_1
LD IX,(n_1n_2)	DD 2A n_2n_1
LD IX,n_1n_2	DD 21 n_2n_1
LD IY,(n_1n_2)	FD 2A n_2n_1
LD IY,n_1n_2	FD 21 n_2n_1
LD SP,(n_1n_2)	ED 7B n_2n_1
LD SP,HL	F9
LD SP,IX	DD F9
LD SP,IY	FD F9
LD SP,n_1n_2	31 n_2n_1
サブルーチン呼び出し命令	
RET C	D8
RET M	F8
RET NC	D0
RET NZ	C0
RET P	F0
RET PE	E8
RET P0	E0
RET Z	C8
RET	C9
8ビット加算命令	
SBC A,(HL)	9E
SBC A,(IX+d)	DD 9E d
SBC A,(IY+d)	FD 9E d
SBC A,A	9F
SBC A,B	98
SBC A,C	99
SBC A,D	9A
SBC A,E	9B
SBC A,H	9C
SBC A,L	9D
SBC A,n	DE n
SUB (HL)	96
SUB (IX+d)	DD 96 d
SUB (IY+d)	FD 96 d
SUB A	97
SUB B	90
SUB C	91
SUB D	92
SUB E	93
SUB H	94
SUB n	D6 n
16ビット加算命令	
SBC HL,BC	ED 42
SBC HL,DE	ED 52
SBC HL,HL	ED 62
SBC HL,SP	ED 72

㊟ n_2n_1はn_2を下位バイト，n_1を上位バイトとする16ビットの値です。
インストラクションコードでは，n_2，n_1と2バイトになります。
例　DC n_2n_1はDC，D_2，n_1の3バイト

第3部 6809マシン語入門編



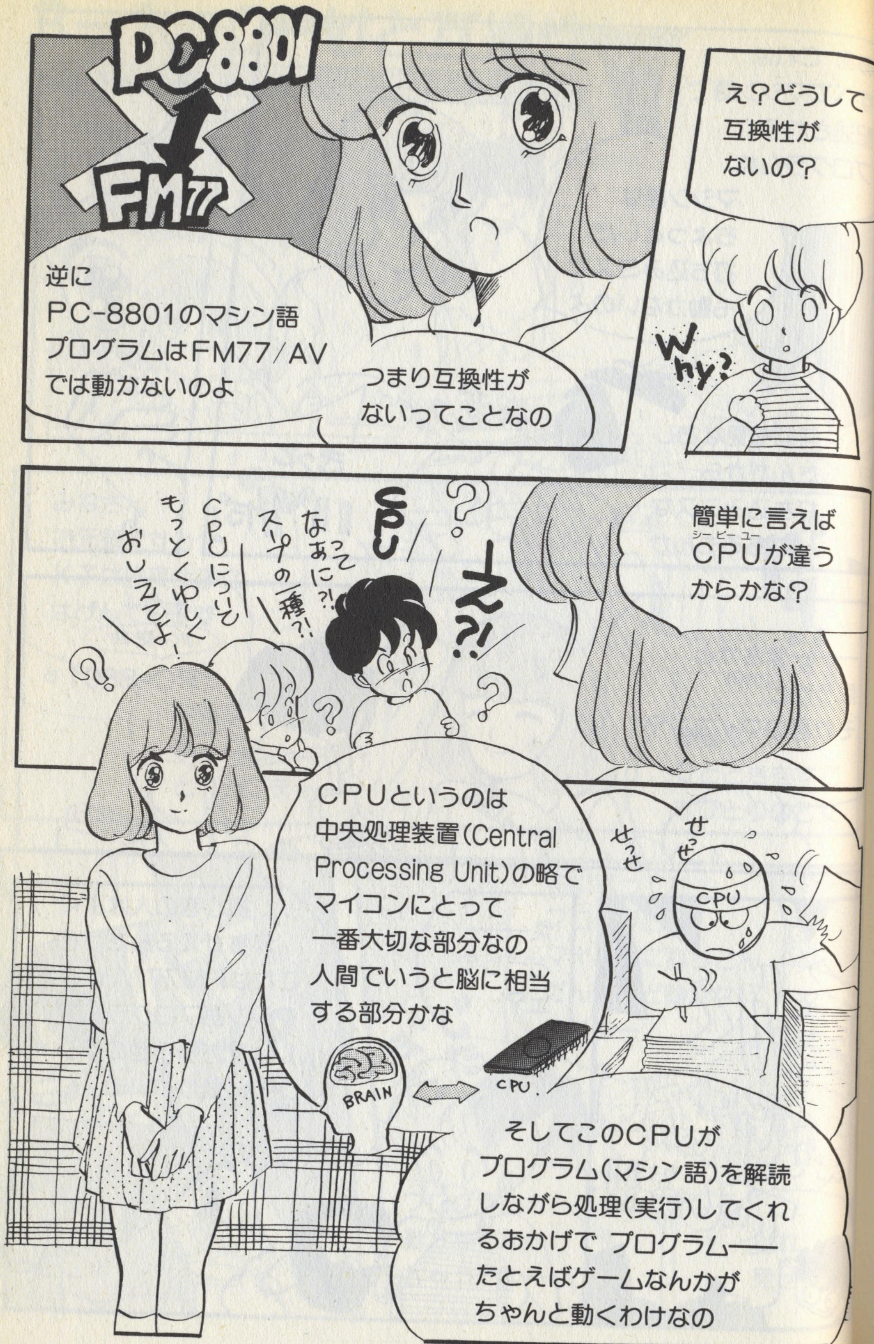
3−1　CPUと互換性



みなさん
お元気ですか?
K子は日夜仕事に
はげんでいます
でも　今日の
午後はひとりで
ゆ――っくり
とハーブティー
優雅な
ひとときをすご
すひとりの乙女
バタン!
K子さぁん
HELP ME!!
本に載ってた
プログラムを
マイコンに打ちこん
だんだけど動かない
んだよー
じたばた
プログラムが
動かなくなっ
ちゃったよー
どれ
見せて

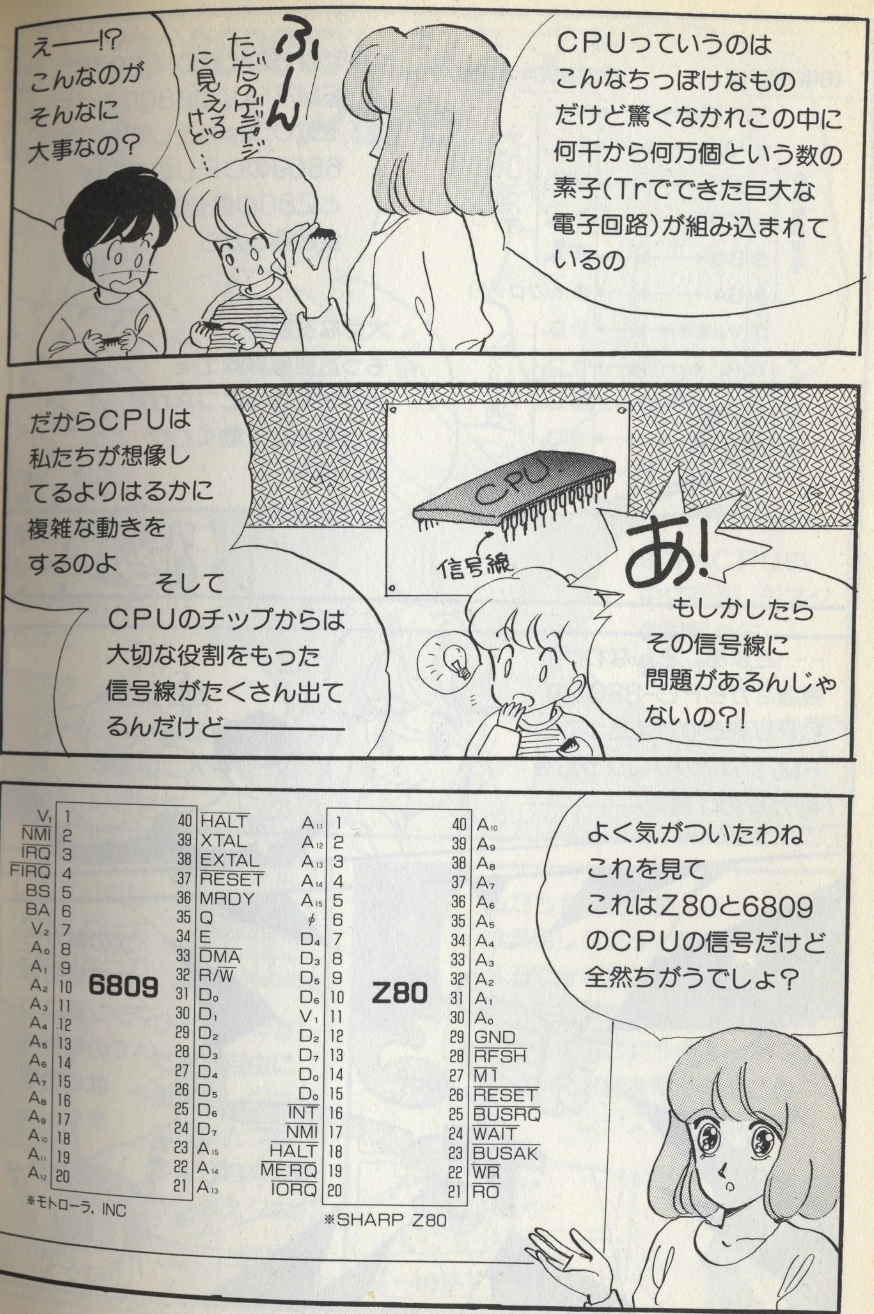


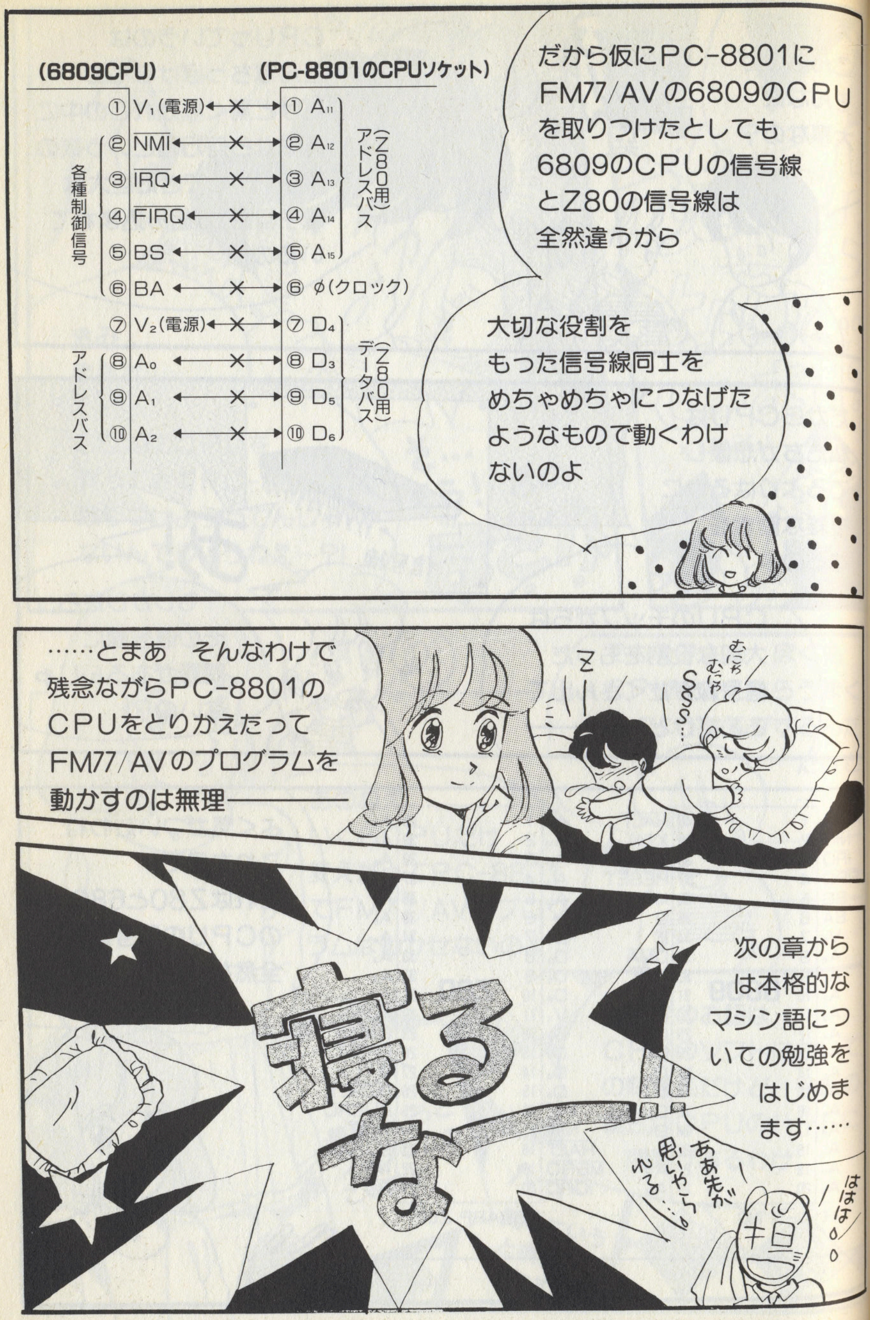
ああ これは
すべてマシン語で
記述される
プログラムね
ほんとにちゃんとチェックした?
マシン語は
ちょっとした
打ち込みミスで
も動かないのよ
何回も見なおし
たんだから
打ち込みミスな
んてあるもんか
きっと
バグだ!
うん?
――まさかと
思うけど……
それ君のマイコン?
なんで動かないんだよ
そう!
ぼくの愛機!
PC-8801
動く
わけない
でしょ!!
え――
どうして
ェ?!
初心者の人はよく
まちがえるんだけど
これはFM77/AV の
マシン語プログラムで
しょ?だから他のマシ
ンじゃ動かないの

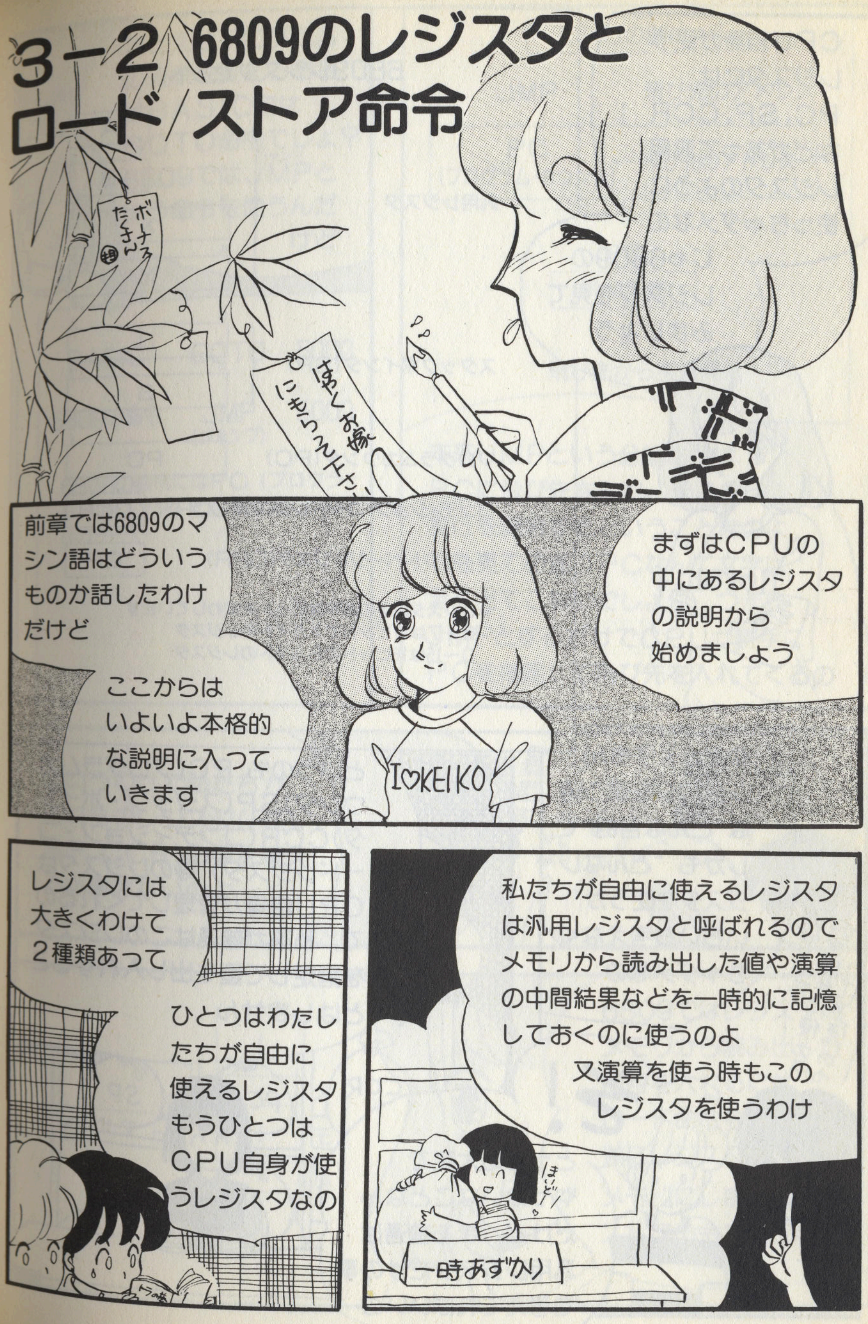

PC8801
FM77
逆に
PC-8801のマシン語
プログラムはFM77/AV
では動かないのよ
つまり互換性が
ないってことなの
え?どうして
互換性が
ないの?
Why?
簡単に言えば
CPU(シーピーユー)が違う
からかな?
CPU
え?!
CPUってなあに?!
スープの一種?!
CPUについてもっとくわしくおしえてよ!
CPUというのは
中央処理装置(Central
Processing Unit)の略で
マイコンにとって
一番大切な部分なの
人間でいうと脳に相当
する部分かな
せせ
せせ
CPU
BRAIN
CPU
そしてこのCPUが
プログラム(マシン語)を解読
しながら処理(実行)してくれ
るおかげで プログラム——
たとえばゲームなんかが
ちゃんと動くわけなの

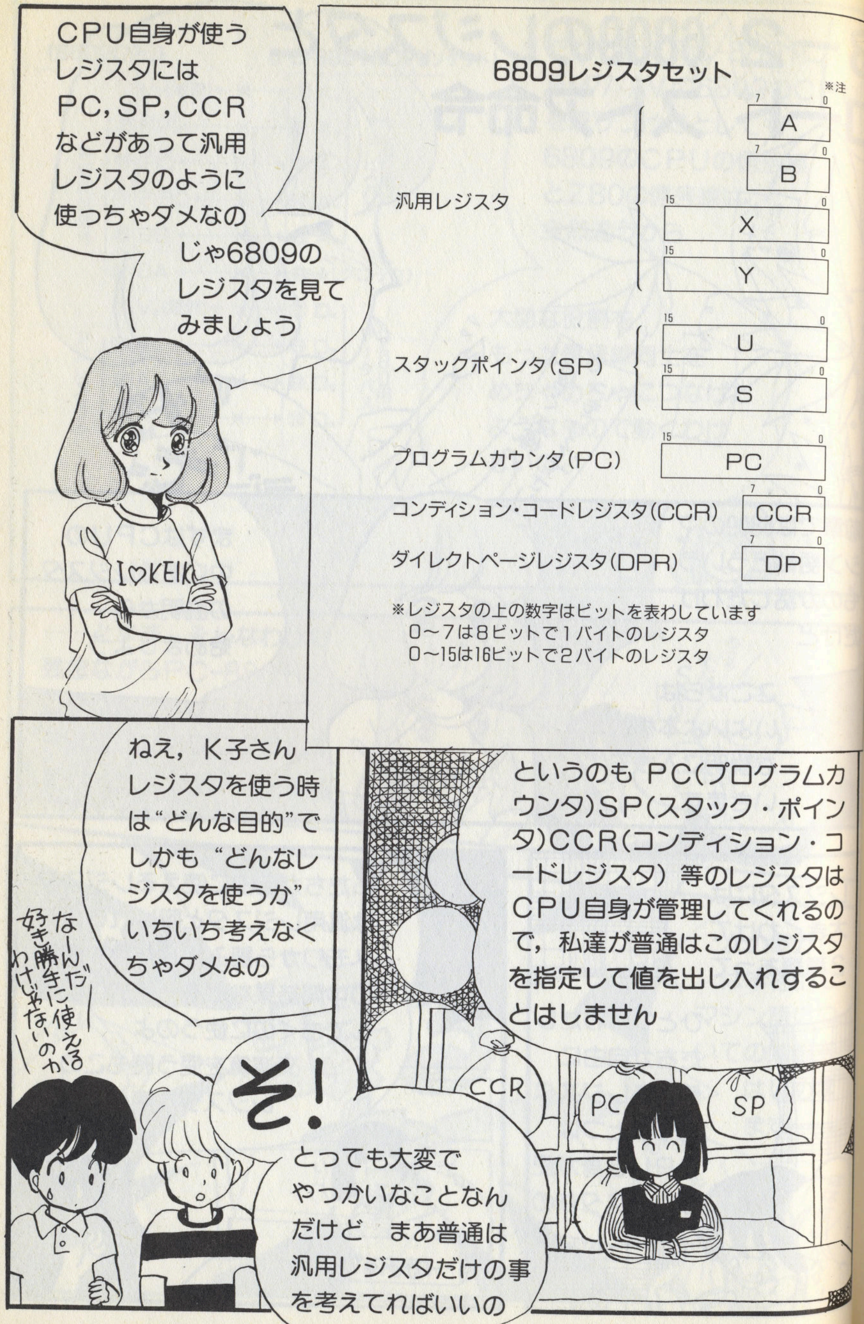

ところが困った事に
CPUは一般には
他のCPUのプログラム
を処理(実行)でき
ないの
つまり
外国人が自分の
国の言葉しか
話せない(理解
できない)よう
なものね
6809
Z-80
だからFM77/AV
(CPUは6809)
のプログラムは
PC-8801(CP
UはZ80)では
動かないのよ
なーんだ
簡単なこと
じゃないか
CPUがちがう
って事なんでしょ?
ぼくはもう
わかったもんね
わかった
って……
何が?
PC-8801で
FM-77/AV
のマシン語プロ
グラムを動かす
方法だよ
コラ!
ぐえ
ちょっと
待った!!

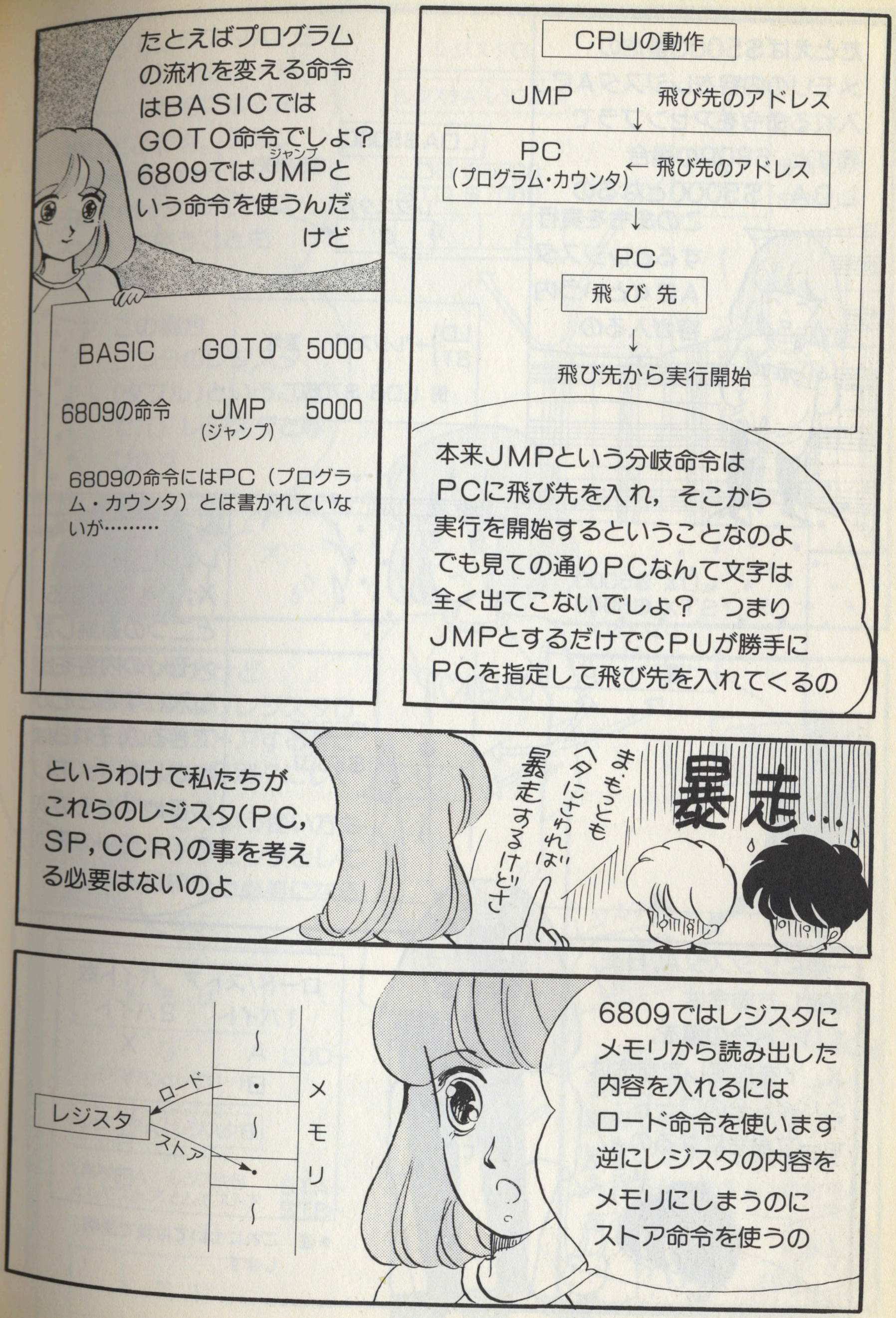

こ、こわい……
きみ まさか……
6809のCPUを 買ってきてPC-8801に 取りつけようってんじゃ ないんでしょうねえ…?!
そ… そう！ その まさか
できるわけ ないでしょぉ!?
そんなこと！
えー?! なんでだよー！
ぶつぶつ
もうほんとに すぐ早合点して……
でもK子さん ぼくも CPUをかえたらうまく いくような気がするんだ だけど ……？
あのねえ いくらCPUだけ 変えたってPC-8801 でFM77/AVのプログ ラムは動かせないのよ
今までのお話は CPUのソフトについて の問題点だけど 今度の 場合はCPUのハードに 問題があるわけ
ソフト 簡単にいうと プログラムの ことです
ハード 簡単にいう と機械のこ とです

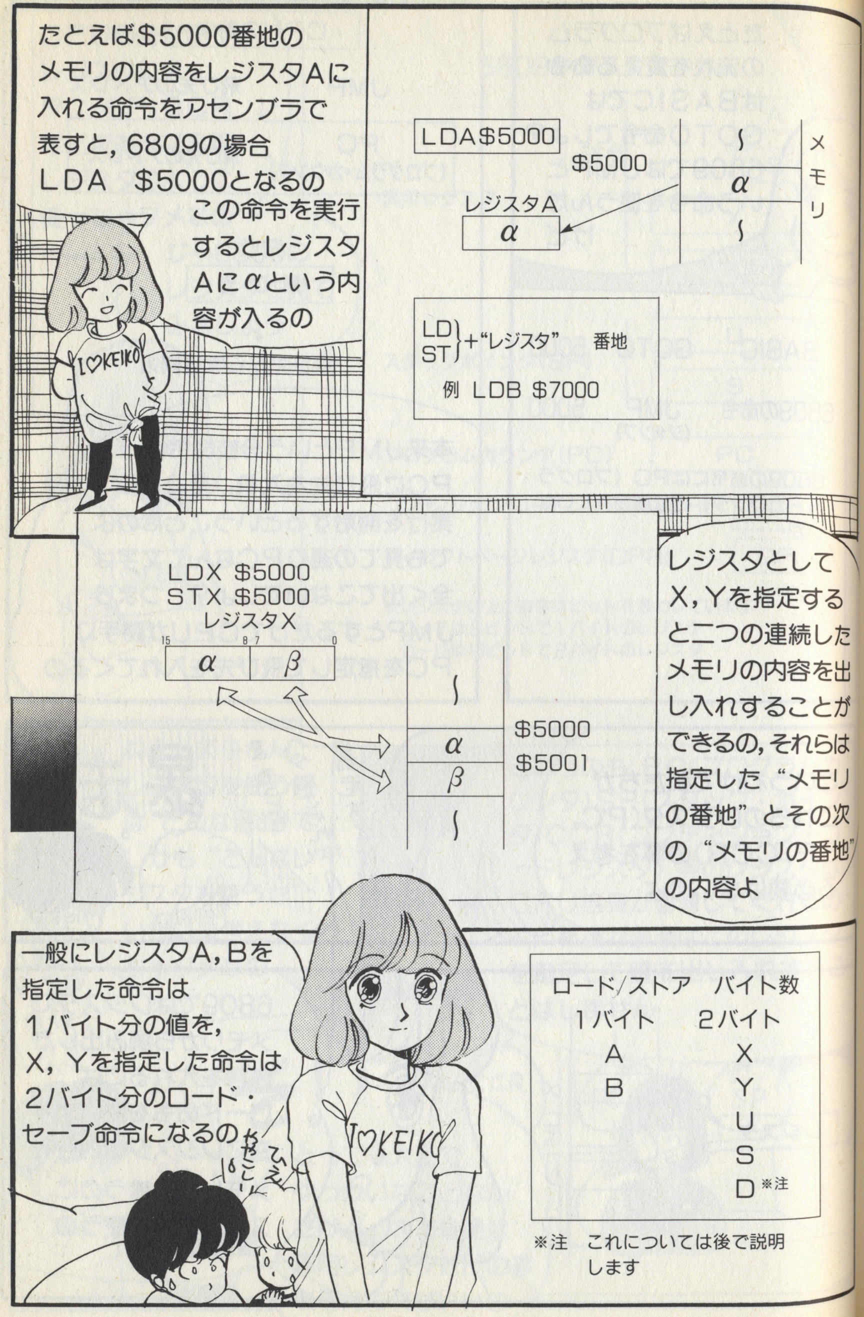

えー!? こんなのが そんなに 大事なの？
ふーん ただのゲジゲジに見えるけど……
CPUっていうのは こんなちっぽけなもの だけど驚くなかれこの中に 何千から何万個という数の 素子(Trでできた巨大な 電子回路)が組み込まれて いるの
だからCPUは 私たちが想像し てるよりはるかに 複雑な動きを するのよ そして CPUのチップからは 大切な役割をもった 信号線がたくさん出て るんだけど――
CPU
信号線
あ！ もしかしたら その信号線に 問題があるんじゃ ないの?!
6809
1 V_1, 2 $\overline{NMI}$, 3 $\overline{IRQ}$, 4 $\overline{FIRQ}$, 5 BS, 6 BA, 7 V_2, 8 A_0, 9 A_1, 10 A_2, 11 A_3, 12 A_4, 13 A_5, 14 A_6, 15 A_7, 16 A_8, 17 A_9, 18 A_{10}, 19 A_{11}, 20 A_{12}
40 $\overline{HALT}$, 39 XTAL, 38 EXTAL, 37 $\overline{RESET}$, 36 MRDY, 35 Q, 34 E, 33 $\overline{DMA}$, 32 $R/\overline{W}$, 31 D_0, 30 D_1, 29 D_2, 28 D_3, 27 D_4, 26 D_5, 25 D_6, 24 D_7, 23 A_{15}, 22 A_{14}, 21 A_{13}
※モトローラ, INC
Z80
1 A_{11}, 2 A_{12}, 3 A_{13}, 4 A_{14}, 5 A_{15}, 6 ϕ, 7 D_4, 8 D_3, 9 D_5, 10 D_6, 11 V_1, 12 D_2, 13 D_7, 14 D_0, 15 D_0, 16 $\overline{INT}$, 17 $\overline{NMI}$, 18 $\overline{HALT}$, 19 $\overline{MERQ}$, 20 $\overline{IORQ}$
40 A_{10}, 39 A_9, 38 A_8, 37 A_7, 36 A_6, 35 A_5, 34 A_4, 33 A_3, 32 A_2, 31 A_1, 30 A_0, 29 GND, 28 $\overline{RFSH}$, 27 $\overline{M1}$, 26 $\overline{RESET}$, 25 $\overline{BUSRQ}$, 24 $\overline{WAIT}$, 23 $\overline{BUSAK}$, 22 $\overline{WR}$, 21 $\overline{RO}$
※SHARP Z80
よく気がついたわね これを見て これはZ80と6809 のCPUの信号だけど 全然ちがうでしょ？

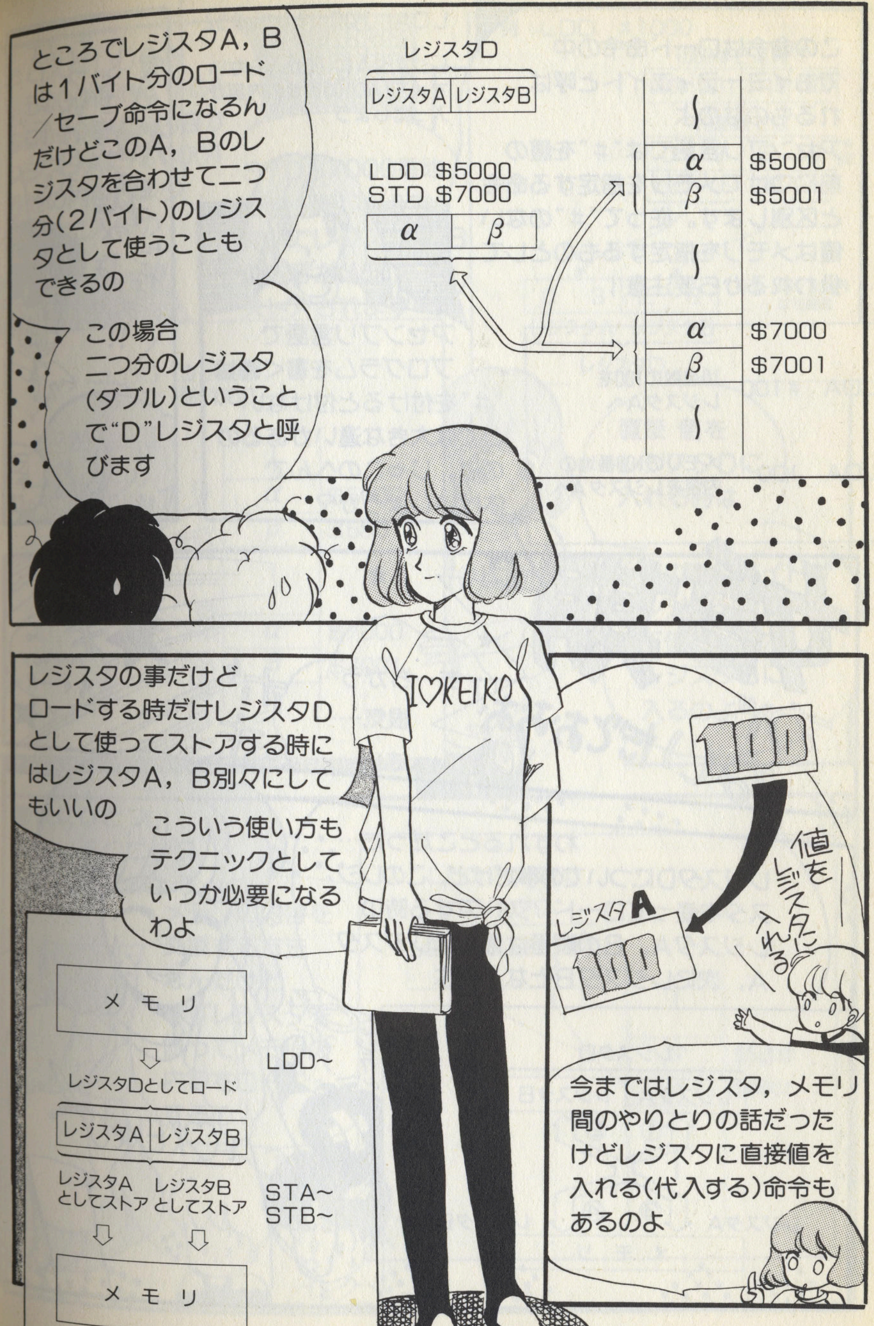

(6809CPU)
(PC-8801のCPUソケット)
① V1(電源) — ① A11
② NMI — ② A12
③ IRQ — ③ A13
④ FIRQ — ④ A14
⑤ BS — ⑤ A15
⑥ BA — ⑥ φ(クロック)
⑦ V2(電源) — ⑦ D4
⑧ A0 — ⑧ D3
⑨ A1 — ⑨ D5
⑩ A2 — ⑩ D6
各種制御信号
アドレスバス
アドレスバス(Z80用)
データバス(Z80用)
だから仮にPC-8801にFM77/AVの6809のCPUを取りつけたとしても6809のCPUの信号線とZ80の信号線は全然違うから
大切な役割をもった信号線同士をめちゃめちゃにつなげたようなもので動くわけないのよ
……とまあ そんなわけで残念ながらPC-8801のCPUをとりかえたってFM77/AVのプログラムを動かすのは無理――
次の章からは本格的なマシン語についての勉強をはじめます……
寝るなー!!
ああ先が思いやられる……
ははは。

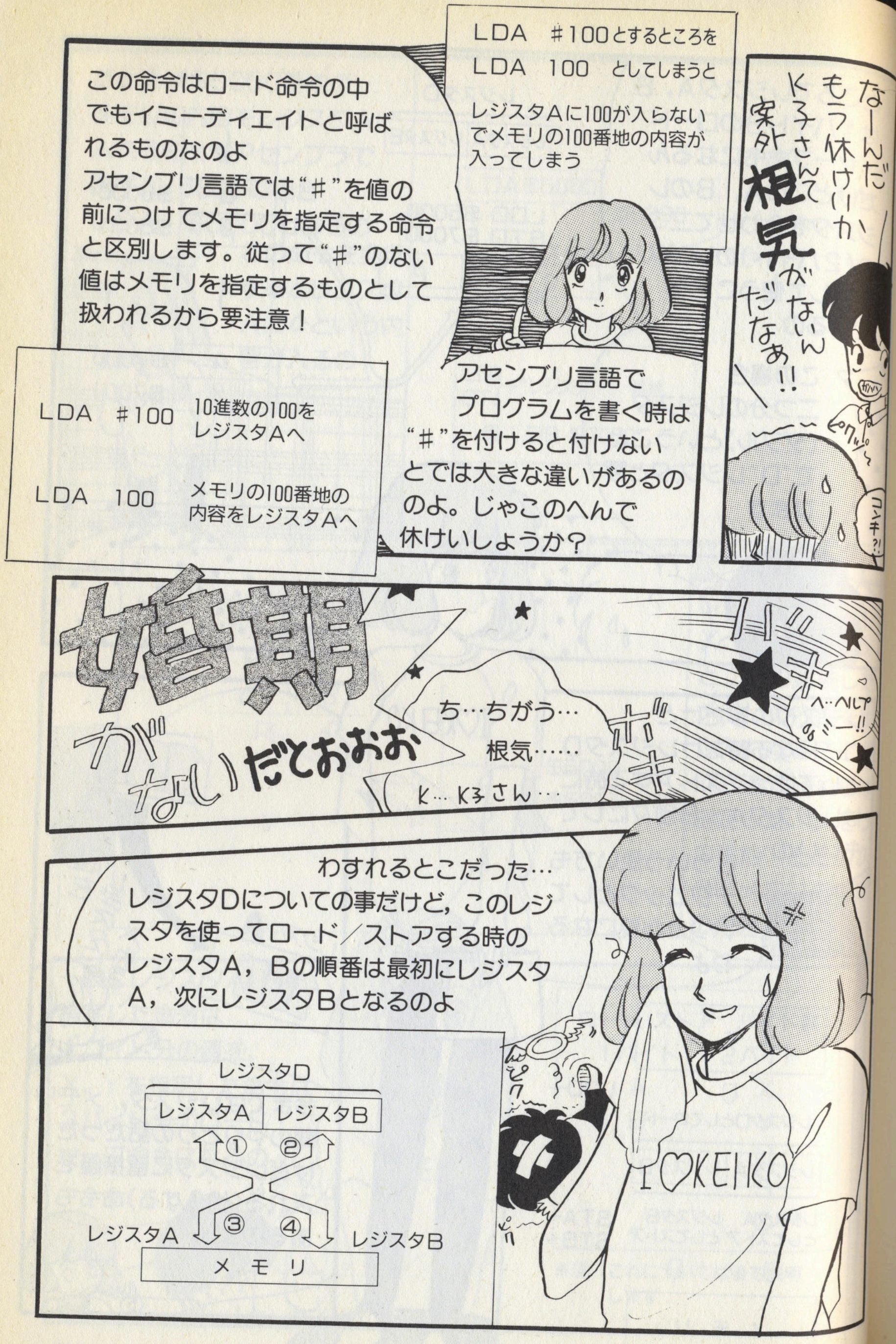

3-2 6809のレジスタとロード/ストア命令
前章では6809のマシン語はどういうものか話したわけだけど
ここからはいよいよ本格的な説明に入っていきます
まずはCPUの中にあるレジスタの説明から始めましょう
I♡KEIKO
レジスタには大きくわけて2種類あって
ひとつはわたしたちが自由に使えるレジスタもうひとつはCPU自身が使うレジスタなの
私たちが自由に使えるレジスタは汎用レジスタと呼ばれるのでメモリから読み出した値や演算の中間結果などを一時的に記憶しておくのに使うのよ
又演算を使う時もこのレジスタを使うわけ
一時あずかり

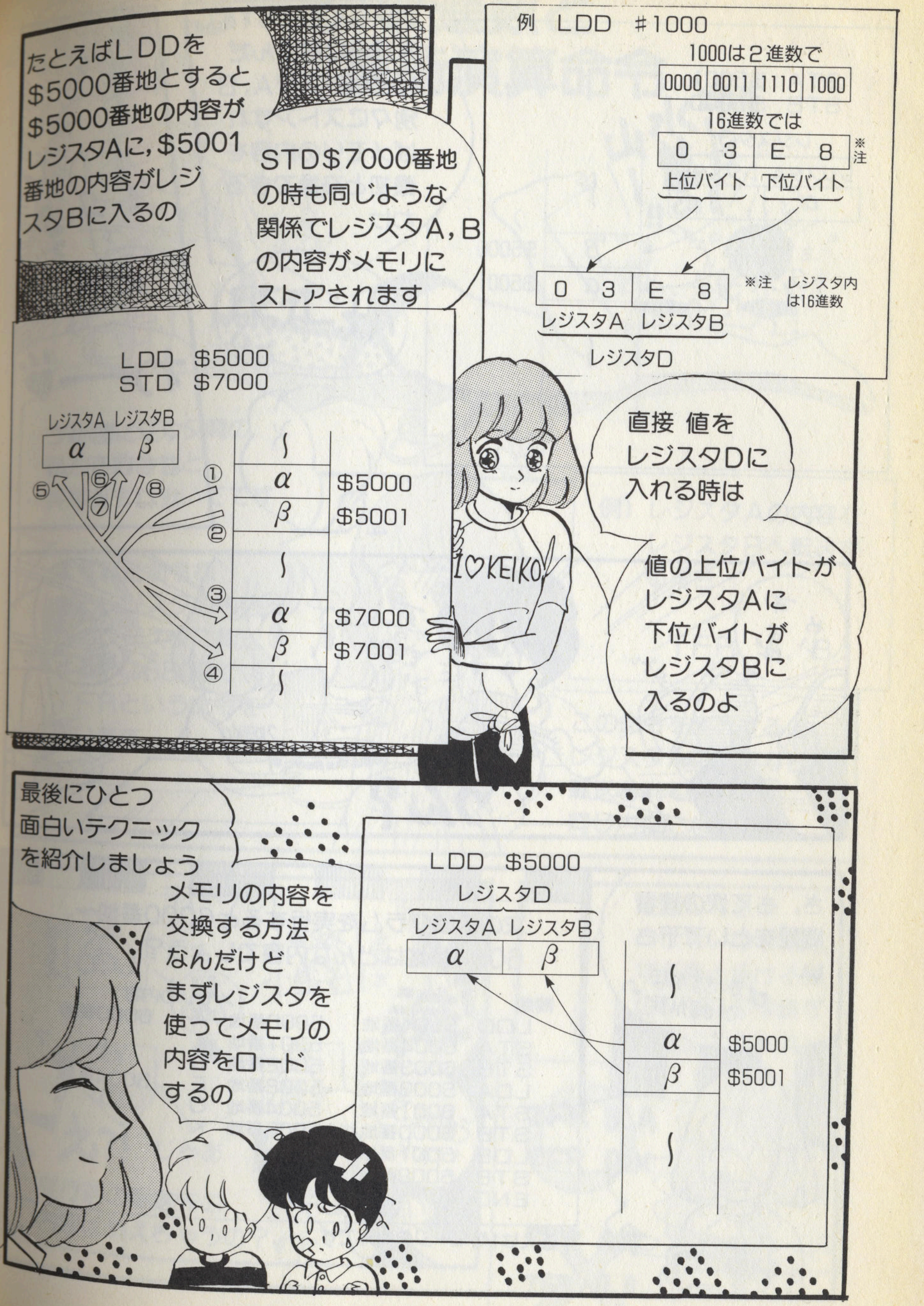

CPU自身が使うレジスタにはPC, SP, CCRなどがあって汎用レジスタのように使っちゃダメなの
じゃ6809のレジスタを見てみましょう
6809レジスタセット
汎用レジスタ
A
B
X
Y
スタックポインタ(SP)
U
S
プログラムカウンタ(PC)
PC
コンディション・コードレジスタ(CCR)
CCR
ダイレクトページレジスタ(DPR)
DP
※注
※レジスタの上の数字はビットを表わしています
0～7は8ビットで1バイトのレジスタ
0～15は16ビットで2バイトのレジスタ
I♡KEIKO
ねえ, K子さん レジスタを使う時は"どんな目的"でしかも"どんなレジスタを使うか"いちいち考えなくちゃダメなの
なーんだ 好き勝手に使えるわけじゃないのか
そ!
とっても大変でやっかいなことなんだけど まあ普通は汎用レジスタだけの事を考えてればいいの
というのも PC(プログラムカウンタ)SP(スタック・ポインタ)CCR(コンディション・コードレジスタ) 等のレジスタはCPU自身が管理してくれるので, 私達が普通はこのレジスタを指定して値を出し入れすることはしません
CCR
PC
SP


たとえばプログラムの流れを変える命令はBASICではGOTO命令でしょ? 6809ではJMP(ジャンプ)という命令を使うんだけど
BASIC GOTO 5000
6809の命令 JMP(ジャンプ) 5000
6809の命令にはPC（プログラム・カウンタ）とは書かれていないが………
CPUの動作
JMP 飛び先のアドレス
PC (プログラム・カウンタ) ← 飛び先のアドレス
PC
飛び先
飛び先から実行開始
本来JMPという分岐命令はPCに飛び先を入れ, そこから実行を開始するということなのよ でも見ての通りPCなんて文字は全く出てこないでしょ? つまりJMPとするだけでCPUが勝手にPCを指定して飛び先を入れてくるの
というわけで私たちがこれらのレジスタ(PC, SP, CCR)の事を考える必要はないのよ
ま, もっとも ヘタにさわれば暴走するけどさ
暴走…
6809ではレジスタにメモリから読み出した内容を入れるにはロード命令を使います 逆にレジスタの内容をメモリにしまうのにストア命令を使うの
レジスタ
ロード
ストア
メモリ


たとえば$5000番地のメモリの内容をレジスタAに入れる命令をアセンブラで表すと，6809の場合 LDA　$5000となるの
この命令を実行するとレジスタAにαという内容が入るの
LDA$5000
$5000
α
レジスタA
α
メモリ
LD
ST
+"レジスタ"　番地
例 LDB $7000
LDX $5000
STX $5000
レジスタX
15
8 7
0
α
β
$5000
$5001
α
β
レジスタとしてX，Yを指定すると二つの連続したメモリの内容を出し入れすることができるの，それらは指定した"メモリの番地"とその次の"メモリの番地"の内容よ
一般にレジスタA，Bを指定した命令は1バイト分の値を，X，Yを指定した命令は2バイト分のロード・セーブ命令になるのね
ロード/ストア　バイト数
1バイト　2バイト
A　X
B　Y
U
S
D※注
※注　これについては後で説明します

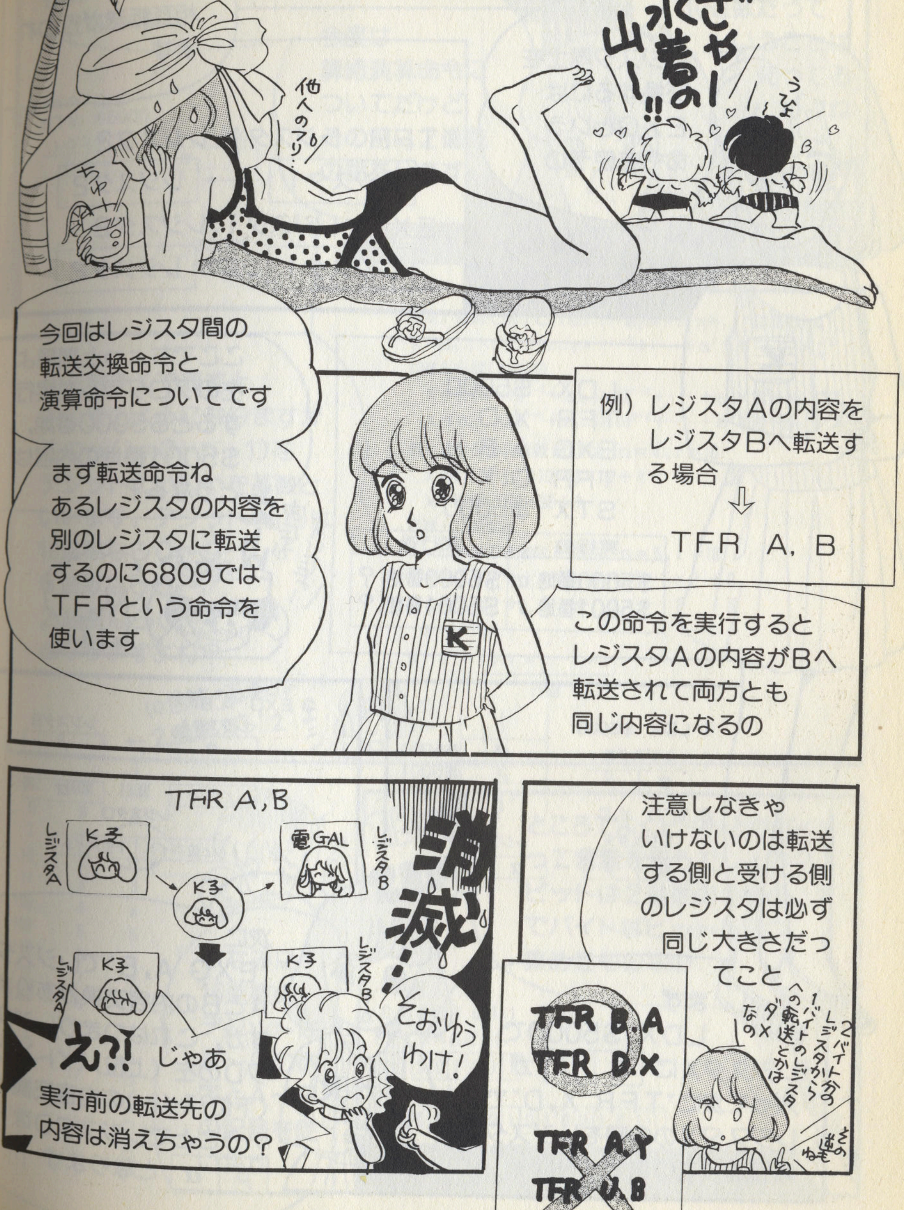

ところでレジスタA，Bは1バイト分のロード/セーブ命令になるんだけどこのA，Bのレジスタを合わせて二つ分(2バイト)のレジスタとして使うこともできるの
レジスタD
レジスタA　レジスタB
LDD $5000
STD $7000
α　β
α　$5000
β　$5001
α　$7000
β　$7001
この場合二つ分のレジスタ(ダブル)ということで"D"レジスタと呼びます
レジスタの事だけどロードする時だけレジスタDとして使ってストアする時にはレジスタA，B別々にしてもいいの
こういう使い方もテクニックとしていつか必要になるわよ
メ モ リ
レジスタDとしてロード
LDD~
レジスタA　レジスタB
レジスタAとしてストア　レジスタBとしてストア
STA~
STB~
メ モ リ
I♡KEIKO
100
レジスタA
100
値をレジスタに入れる
今まではレジスタ，メモリ間のやりとりの話だったけどレジスタに直接値を入れる(代入する)命令もあるのよ


なーんだ
もう休けいか
K子さんて
案外根気がないんだなぁ!!
LDA #100とするところを
LDA 100 としてしまうと
レジスタAに100が入らないでメモリの100番地の内容が入ってしまう
アセンブリ言語でプログラムを書く時は"#"を付けると付けないとでは大きな違いがあるのよ。じゃこのへんで休けいしょうか?
この命令はロード命令の中でもイミーディエイトと呼ばれるものなのよ
アセンブリ言語では"#"を値の前につけてメモリを指定する命令と区別します。従って"#"のない値はメモリを指定するものとして扱われるから要注意!
LDA #100 10進数の100をレジスタAへ
LDA 100 メモリの100番地の内容をレジスタAへ
へ…ヘルプミー!!
婚期がないだとおおお
ち…ちがう…
根気……
K…K子さん…
わすれるとこだった…
レジスタDについての事だけど、このレジスタを使ってロード/ストアする時のレジスタA，Bの順番は最初にレジスタA，次にレジスタBとなるのよ
レジスタD
レジスタA レジスタB
① ②
③ ④
レジスタA レジスタB
メ モ リ
I♡KEIKO


例 LDD #1000
1000は2進数で
0000 0011 1110 1000
16進数では
0 3 E 8 ※注
上位バイト 下位バイト
0 3 E 8
※注 レジスタ内は16進数
レジスタA レジスタB
レジスタD
たとえばLDDを$5000番地とすると$5000番地の内容がレジスタAに,$5001番地の内容がレジスタBに入るの
STD$7000番地の時も同じような関係でレジスタA，Bの内容がメモリにストアされます
LDD $5000
STD $7000
レジスタA レジスタB
α β
① ② ③ ④ ⑤ ⑥ ⑦ ⑧
α $5000
β $5001
α $7000
β $7001
直接 値をレジスタDに入れる時は
値の上位バイトがレジスタAに下位バイトがレジスタBに入るのよ
I♡KEIKO
最後にひとつ面白いテクニックを紹介しましょう
メモリの内容を交換する方法なんだけどまずレジスタを使ってメモリの内容をロードするの
LDD $5000
レジスタD
レジスタA レジスタB
α β
α $5000
β $5001



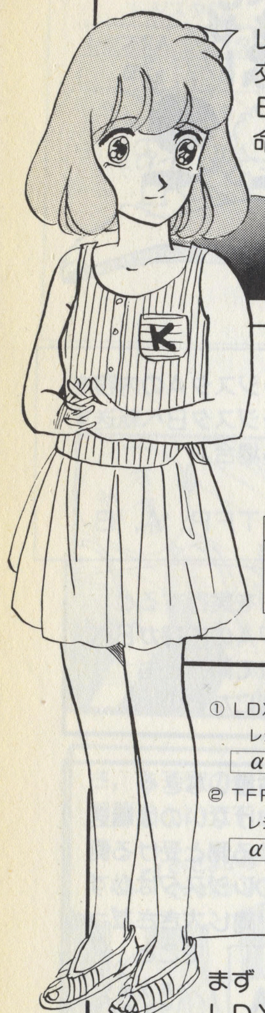
3−3　2進数と算術演算命令



じゃあ次は
レジスタ交換命令に
ついて話すわね

EXG レジスタ1,レジスタ2
例 EXG D,X

この命令は
TFRとちがって
レジスタ間で
相互転送が行われ
ます



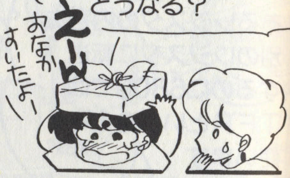
レジスタ同士を
交換するには
EXGという
命令を使うの

TFR レジスタ1,レジスタ2
レジスタ1 ⟶ レジスタ2
EXG レジスタ1,レジスタ2
レジスタ1 ⇄ レジスタ2

```
LDX  $5000
TFR  X,D
EXG  A,B
TFR  D,X
STX  $5000
```

実行前	実行後
$5000番地 α	$5000番地 ?
$5001番地 β	$5001番地 ?

ここでちょっと問題よ
左のプログラムを実行
すると$5000番地,
$5001番地の内容は
どうなる?

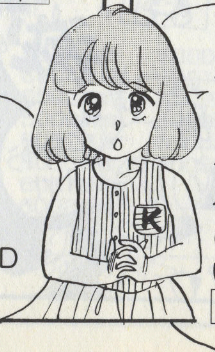


① LDX $5000
レジスタX: α β
α $5000
β $5001

② TFR X,D
レジスタX: α β → レジスタD: α β

③ EXG A,B
レジスタA: α ⇄ レジスタB: β
↓
β α
レジスタD
上位バイト 下位バイト

まず
LDX $5000で
レジスタXに α β が
入って次に"TFR X,D"で
レジスタXの内容をレジスタD
に転送しています



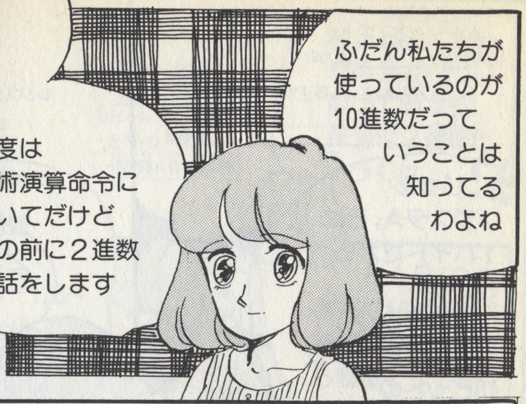
次に
"EXG A,B"でレジスタ
AとBの内容交換がありま
すが,これはつまりレジス
タDの左(上位バイト)と
(下位バイト)の内容交換と
いうことでレジスタDは
β α となります

 ●144ページの答え● 6000番地よりK,E,I,K,O

そしてレジスタDをレジスタ
Xに転送し最後にレジスタX
を$5000番地にしまってい
ます。よって答えは$5000
番地=β, $5001番地=α
となるの

④TFR D,X
STX $5000
レジスタD: β α → レジスタX: α β
β $5000
α $5001

今度は
算術演算命令に
ついてだけど
その前に2進数
の話をします



ふだん私たちが
使っているのが
10進数だって
いうことは
知ってる
わよね

10進数をXで表すと
右の表のようになりますが
右のai(i=0〜n−1)を
デジットといいrを基数と
いいます。デジット(ai)は
10進数なら0〜9
8進数なら0〜7
2進数なら0〜1
となりますよね?

xをr進数で表すと(a_{n-1}〜a_0はデジット,rは基数)

$x=a_{n-1}\times r^{n-1}+a_{n-2}\times r^{n-2}+\cdots\cdots+a_1\times r^1+a_0\times r^0$

10進ならば(r=10)

$x=a_{n-1}\times 10^{n-1}+a_{n-2}\times 10^{n-2}+\cdots\cdots+a_1\times 10^1+a_0\times 10^0$

$\therefore x=a_{n-1}\times 10^{n-1}+a_{n-2}\times 10^{n-2}+\cdots\cdots+a_1\times 10+a_0$

例 4ケタの10進数5731は n=4,r=10

$x=5\times 10^3+7\times 10^2+3\times 10^1+1$

$x=5000+700+30+1$

2進→10進変換

5ケタの2進数11010は n=5,r=10

$x=1\times 2^4+1\times 2^3+0\times 2^2+1\times 2^1+0$

$x=16+8+0+2+0$

$x=26$

※ $r^n=\underbrace{r\times r\times\cdots\cdots r\times r}_{n回}$



2進	10進	16進
0	0	0
1	1	1
10	2	2
11	3	3
100	4	4
101	5	5
110	6	6
111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

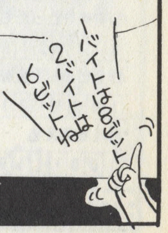


ちなみに
16進数の
デジットは
0〜9の数
字とA〜F
の記号を使
います

1B=8
2B=16

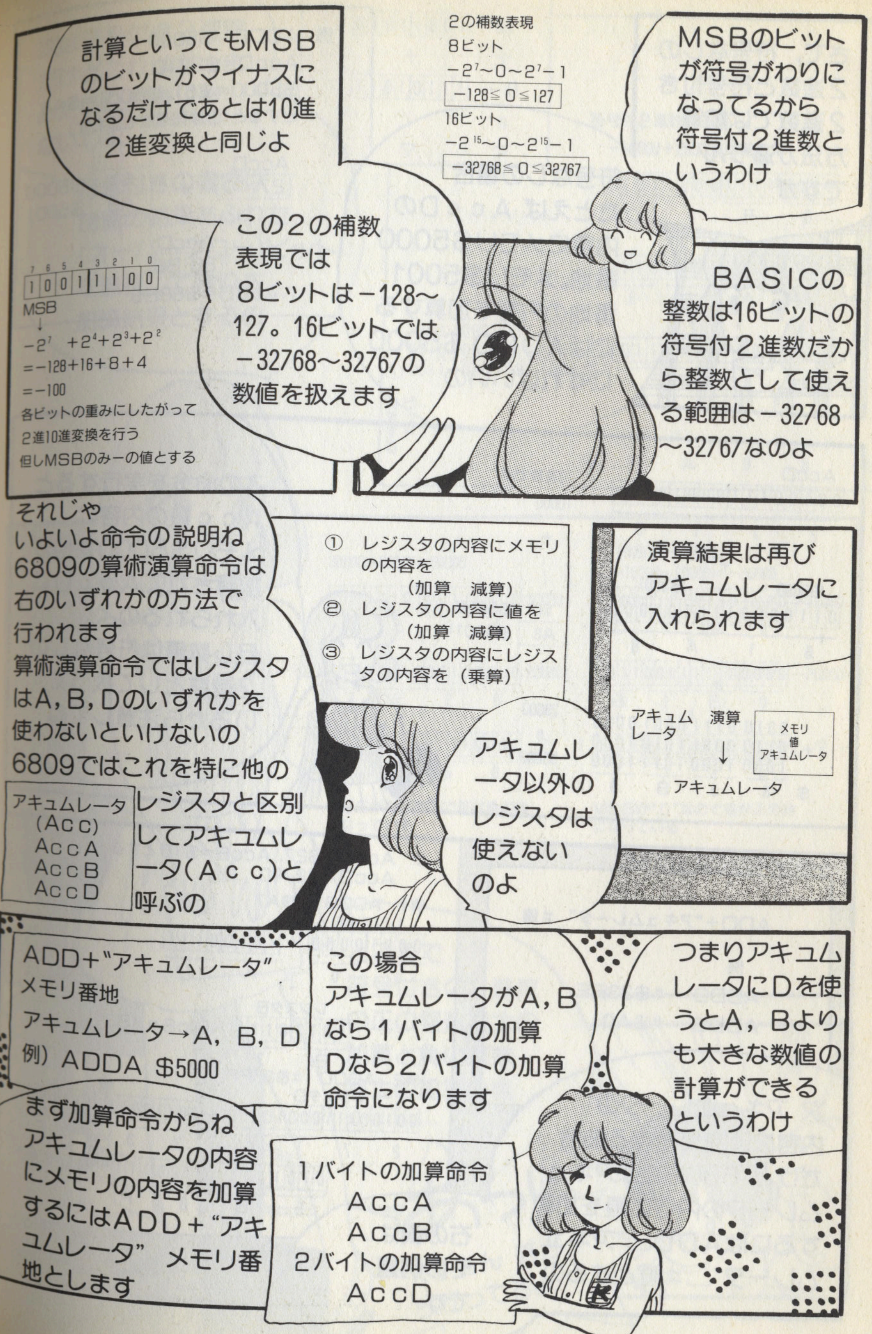
ところでよくビットとかバイト
って言葉を使うでしょう?
ビットは2進数の1桁のこと
でバイトはビットを八つ
集めたものなの

2進数
01001011
1バイト=8ビット
0 1 0 0 1 0 1 1
2バイト=16ビット
0 1 0 0 1 1 0 1 0 1 0 0 1 0 1 1

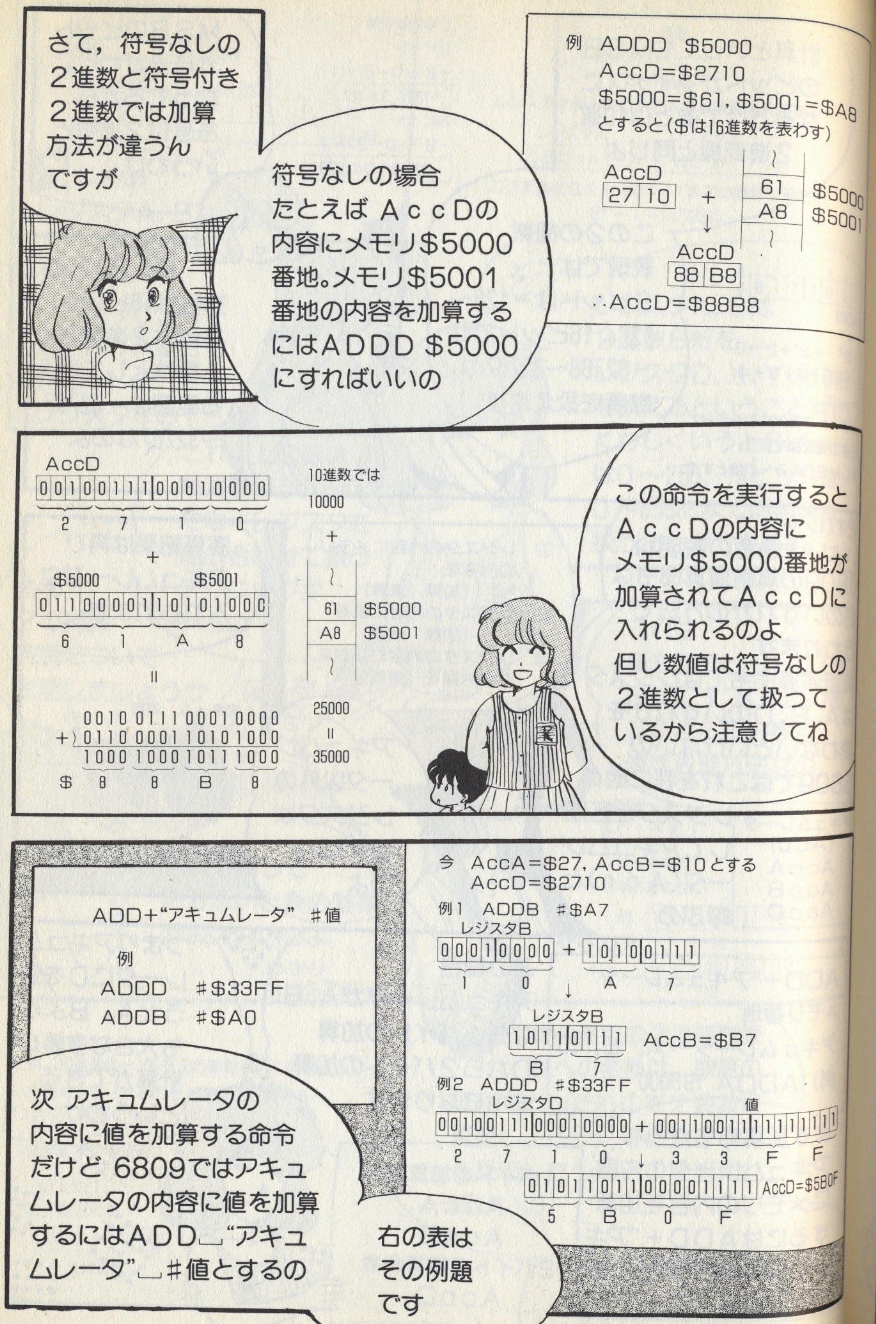




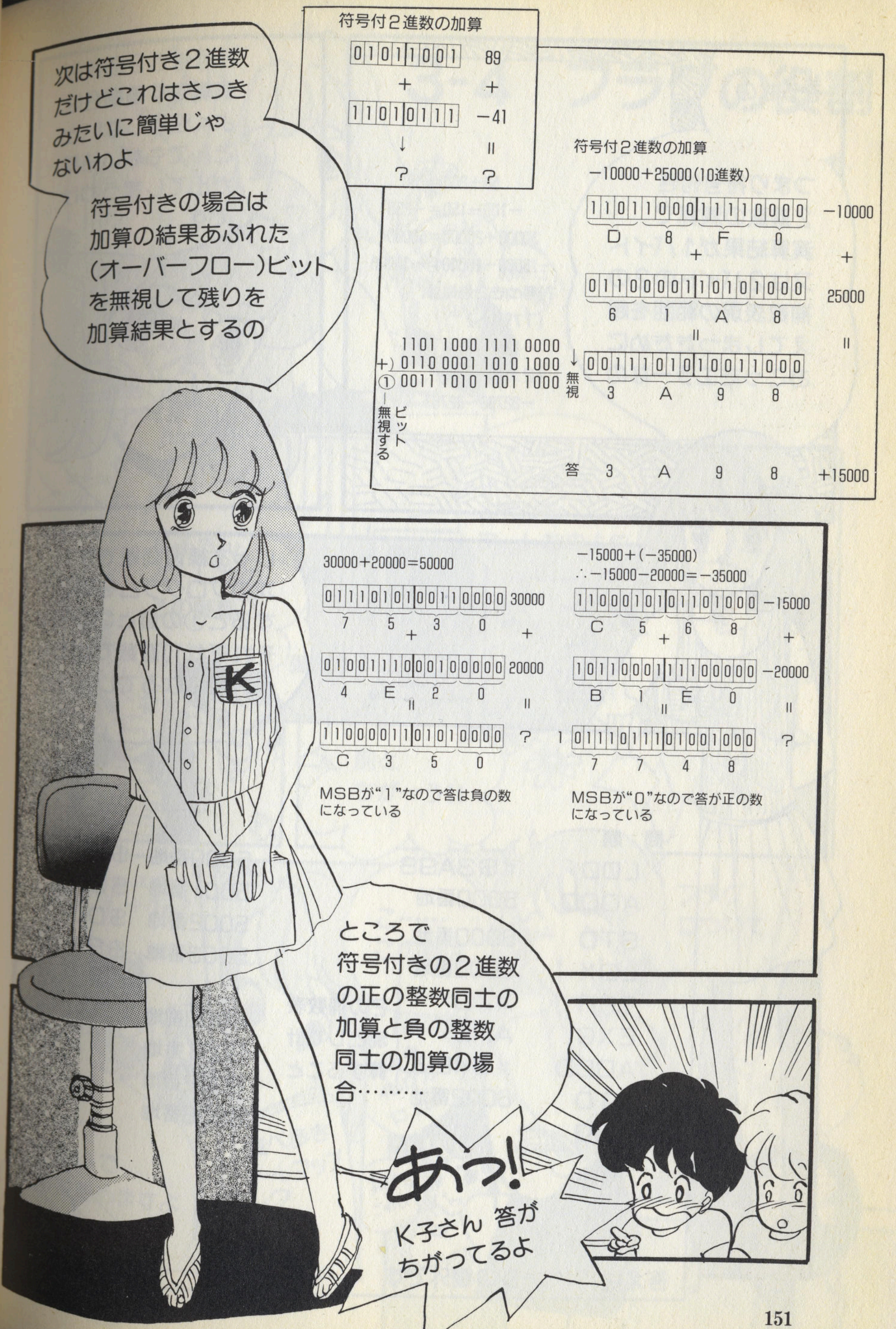
レジスタA，B
MSB LSB
8桁
レジスタD，X，Y，U，S，PC
MSB LSB
16桁
8ビット→2⁸−1
0〜255
16ビット→2¹⁶−1
0〜65535
レジスタを組み合わせて大きな数値を実現する
レジスタA レジスタX
8ビット 16ビット
こうすると0≦x≦2²⁴−1までの数値を扱える
2²⁴−1＝16777215
じゃあ レジスタA，Bは 1バイトだから 8ビット レジスタD，X，Y は2バイトだから 16ビットの レジスタなの？
そう！
レジスタA，Bは 2進数8桁を レジスタD，X，Yは16桁 を扱えるということなの くわしくいうと8ビット は0〜255，16ビットは 0〜65535までの数値を 扱うことができるの
これ以上の数は 単独のレジスタ でなくレジスタを 組み合わせて扱う テクニックが いるの
さあ 2の補数 表現について 説明しましょうか
だんだん わかんなく なってきたぞ
これは 最も左のビット（MS B）が“1”なら負の数 “0”なら正の数を 表す表現方法なの
MSB
1なら負の数
0なら正の数
例 11110110
−10
00001010
10
これを知って おかないと負の整数 が扱えないのよ
しっかり おぼえてね
例 10進数の−100を2の補数表現に変換する
①符号をとったものを2進数に変換する
01100100
②全ビットを反転
10011011
③1を加える
10011100
変換終了
負の整数の2の補数表 現への変換は，数値の 符号をとったものを2進数に 変換し，さらにその全ビット を反転したものに1を加えれ ばいいの
逆に10進数に変換したい 時はちょっとした計算を すればいいの



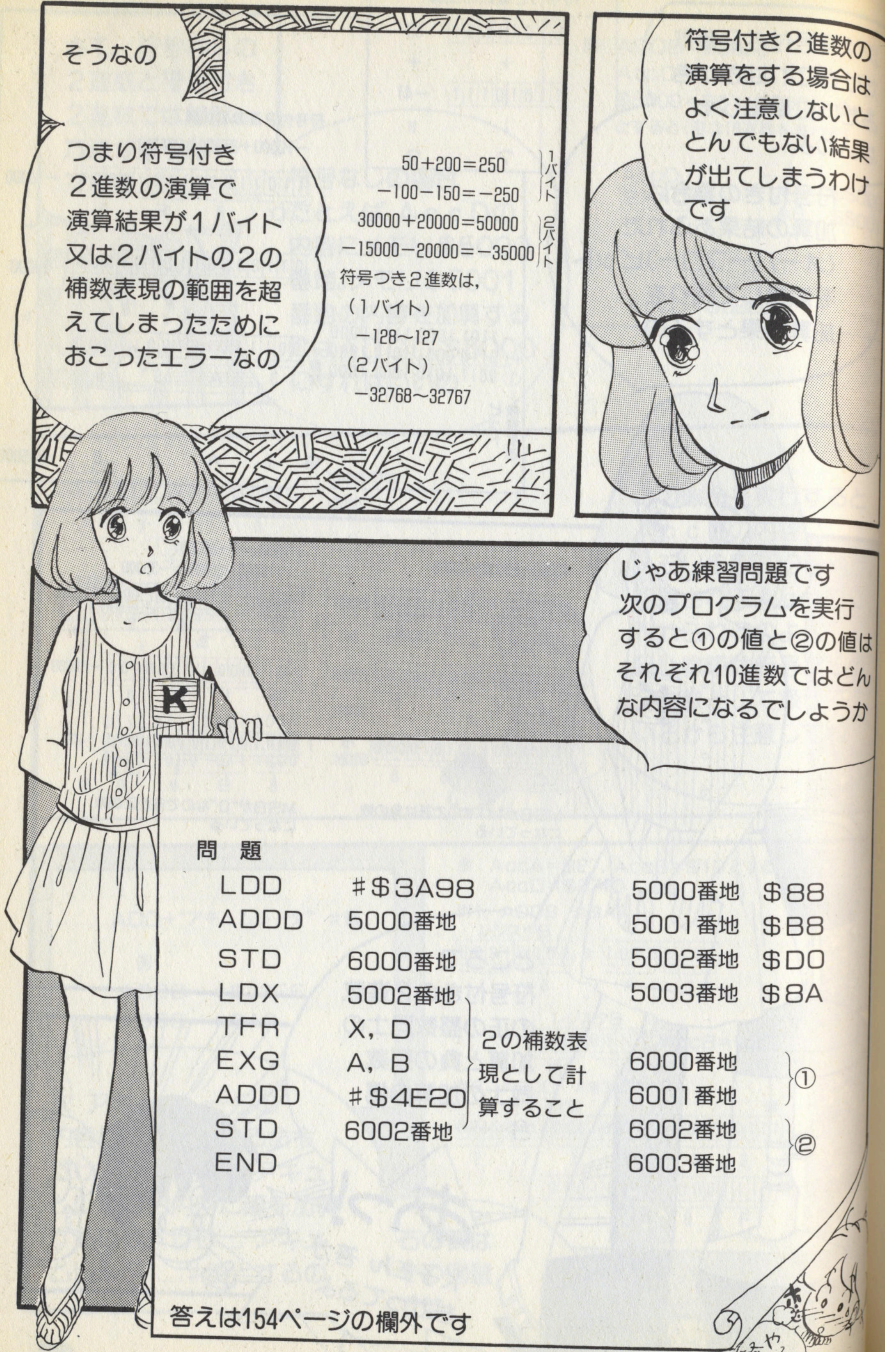
計算といってもMSB のビットがマイナスに なるだけであとは10進 2進変換と同じよ
2の補数表現
8ビット
−2⁷〜0〜2⁷−1
−128≦0≦127
16ビット
−2¹⁵〜0〜2¹⁵−1
−32768≦0≦32767
MSBのビット が符号がわりに なってるから 符号付2進数と いうわけ
7 6 5 4 3 2 1 0
10011100
MSB
−2⁷ ＋2⁴＋2³＋2²
＝−128＋16＋8＋4
＝−100
各ビットの重みにしたがって
2進10進変換を行う
但しMSBのみ一の値とする
この2の補数 表現では 8ビットは−128〜 127。16ビットでは −32768〜32767の 数値を扱えます
BASICの 整数は16ビットの 符号付2進数だか ら整数として使え る範囲は−32768 〜32767なのよ
それじゃ いよいよ命令の説明ね 6809の算術演算命令は 右のいずれかの方法で 行われます 算術演算命令ではレジスタ はA，B，Dのいずれかを 使わないといけないの 6809ではこれを特に他の レジスタと区別 してアキュムレ ータ（Acc）と 呼ぶの
① レジスタの内容にメモリの内容を（加算 減算）
② レジスタの内容に値を（加算 減算）
③ レジスタの内容にレジスタの内容を（乗算）
アキュムレータ（Acc） AccA AccB AccD
演算結果は再び アキュムレータに 入れられます
アキュム レータ 演算 メモリ 値 アキュムレータ
アキュムレータ
アキュムレ ータ以外の レジスタは 使えない のよ
ADD＋“アキュムレータ” メモリ番地
アキュムレータ→A，B，D
例）ADDA $5000
この場合 アキュムレータがA，B なら1バイトの加算 Dなら2バイトの加算 命令になります
つまりアキュム レータにDを使 うとA，Bより も大きな数値の 計算ができる というわけ
まず加算命令からね アキュムレータの内容 にメモリの内容を加算 するにはADD＋“アキ ュムレータ” メモリ番 地とします
1バイトの加算命令 AccA AccB 2バイトの加算命令 AccD



さて，符号なしの2進数と符号付き2進数では加算方法が違うんですが
符号なしの場合 たとえば AccDの内容にメモリ$5000番地。メモリ$5001番地の内容を加算するにはADDD $5000にすればいいの
例 ADDD $5000
AccD=$2710
$5000=$61，$5001=$A8
とすると($は16進数を表わす)
AccD
27 10
+
61 $5000
A8 $5001
AccD
88 B8
∴AccD=$88B8
AccD
0010011100010000
2 7 1 0
+
$5000 $5001
0110000110101000
6 1 A 8
=
0010 0111 0001 0000
+) 0110 0001 1010 1000
1000 1000 1011 1000
$ 8 8 B 8
10進数では
10000
+
25000
=
35000
この命令を実行するとAccDの内容にメモリ$5000番地が加算されてAccDに入れられるのよ 但し数値は符号なしの2進数として扱っているから注意してね
ADD+"アキュムレータ" #値
例
ADDD #$33FF
ADDB #$A0
次 アキュムレータの内容に値を加算する命令だけど 6809ではアキュムレータの内容に値を加算するにはADD␣"アキュムレータ"␣#値とするの
右の表はその例題です
今 AccA=$27，AccB=$10とする
AccD=$2710
例1 ADDB #$A7
レジスタB
00010000 + 10100111
1 0 A 7
レジスタB
10110111 AccB=$B7
B 7
例2 ADDD #$33FF
レジスタD
0010011100010000 + 0011001111111111 値
2 7 1 0 3 3 F F
0101101100001111 AccD=$5B0F
5 B 0 F



次は符号付き2進数だけどこれはさっきみたいに簡単じゃないわよ
符号付2進数の加算
01011001 89
+ +
11010111 −41
↓ =
? ?
符号付きの場合は加算の結果あふれた(オーバーフロー)ビットを無視して残りを加算結果とするの
符号付2進数の加算
−10000+25000(10進数)
1101100011110000 −10000
D 8 F 0
+ +
0110000110101000 25000
6 1 A 8
= =
0011101010011000
3 A 9 8
1101 1000 1111 0000
+) 0110 0001 1010 1000
① 0011 1010 1001 1000
無視
ビット無視する
答 3 A 9 8 +15000
30000+20000=50000
0111010100110000 30000
7 5 3 0
+ +
0100111000100000 20000
4 E 2 0
= =
1100001101010000 ?
C 3 5 0
MSBが"1"なので答は負の数になっている
−15000+(−35000)
∴−15000−20000=−35000
1100010101101000 −15000
C 5 6 8
+ +
1011000111100000 −20000
B 1 E 0
= =
1 0111011101001000 ?
7 7 4 8
MSBが"0"なので答が正の数になっている
ところで符号付きの2進数の正の整数同士の加算と負の整数同士の加算の場合………
あっ!
K子さん 答がちがってるよ



問　題

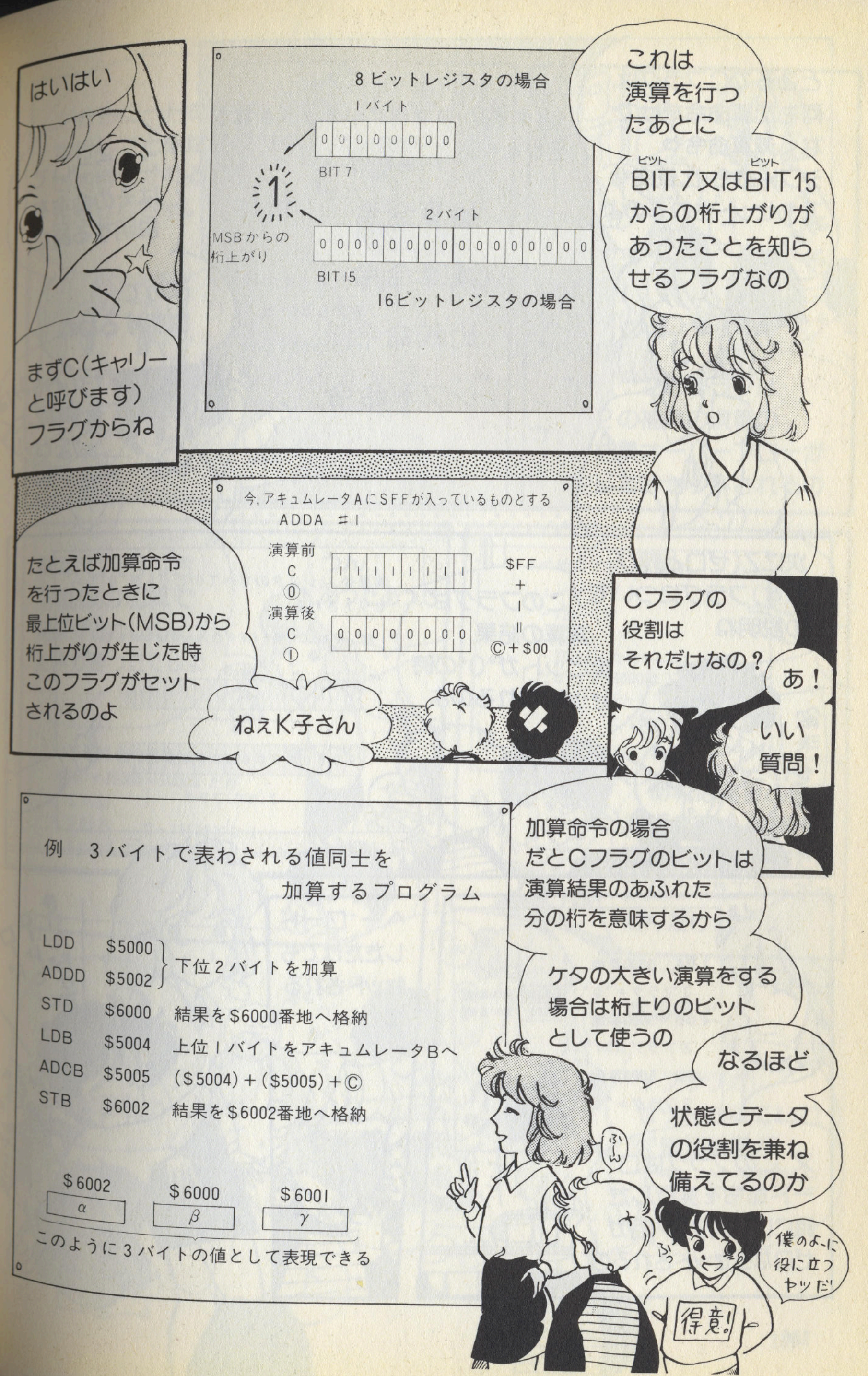
```
LDD   #$3A98          5000番地  $88
ADDD  5000番地         5001番地  $B8
STD   6000番地         5002番地  $D0
LDX   5002番地         5003番地  $8A
TFR   X, D   ┐
EXG   A, B   │2の補数表  6000番地 ┐
ADDD  #$4E20 │現として計  6001番地 ┘①
STD   6002番地┘算すること 6002番地 ┐
END                    6003番地 ┘②
```

答えは154ページの欄外です

3-4　フラグの役割







●152ページの答え● ①50000 ($C350) ②−10000 ($D8F0)



ところでCフラグは何も加算命令だけでなく減算命令やシフト命令・比較命令でもセットされるのよ
C(キャリー)フラグがセットされる主な命令
加算命令
減算命令
比較命令
シフト命令
これについてはその命令を説明する時に
改めて説明するわね
だしおしみ美容によくないッ!!
次にZ(ゼロと呼びます)フラグについての説明ね
このフラグは演算の結果全ビットが"0"の時セットされるのよ
演算後,レジスタのすべてのビットが"0"ならセットされる
例 今AccDに$FFFFが入っているとする
AccD
1111111111111111 +1
0000000000000000
Z(ゼロ)フラグ
解決ゼロ
LDA 5000番地
Zフラグ="0"
LDB 5000番地
Zフラグ="1"
LDD 5000番地
Zフラグ="0"
$77 5000番地
$00 5001番地
又 このフラグはロード命令を実行した時にレジスタの内容がゼロならセットされるの
へー ロードしただけでもセットされるのか
ロードときいただけで
もう!



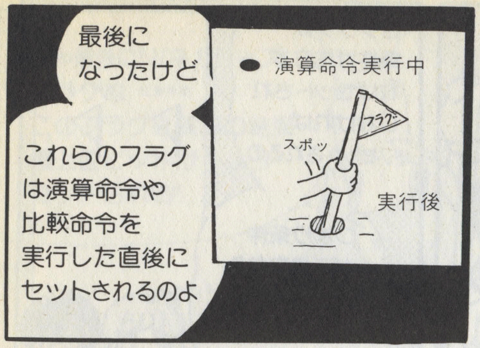
だからロード命令を実行したときはZフラグの内容もかわるから注意しなきゃいけないの
注意一秒 ケガ一生
今次の命令を実行するとZフラグが"1"にセットされる
LDD 5000番地
Zフラグ="1"
$00 5000番地
$00 5001番地
もうあいつは!!
次にV(オーバーフロー)フラグね
このフラグは2の補数の演算の結果オーバーフローが生じた時セットされるの
前章で符号付き2進数の加算で結果がおかしくなった所があったでしょ
うん! あったあった
2の符号付きの演算
30000+20000=50000 だが
$7530
+)$4E20
$C350
で−15536となり答があわない
2の補数で表せる範囲
1バイト
−128〜127
2バイト
−32768〜32767
したがって50000は正の整数の範囲をオーバーしている
∴ V="1"となる
あれは演算の結果数値を表せる範囲をこえてしまったために起こったエラーだったんだけど
いわばそのことを知らせるのがVフラグの役目なの
なるほど
Vフラグがセットされていれば演算結果がエラーだってことだね



じゃ最後にN(ネガティブ)フラグね
このフラグは演算の結果最上位ビットが1ならセットされる
MSBが"1"ならセットされる
10010111
N(ネガティブ)フラグ="1"
01111111
Nフラグ="0"
ざわざわ…
仁義
フィクション
ゲンコーまだですかい?!とっくに期限は…
Nフラグがセットされたからとゆーて何の役に立つんじゃいいい
ちゃんと役目があるんです!!
Nフラグは演算結果が2の補数として正か負かを表わすのよ
2の補数表現では
Nフラグ="1"
負の整数
Nフラグ="0"
正の整数
なるほど
Nフラグがセットされていれば負の整数というぐあいに
そう!
K子さん一見何でもないようなフラグでも大事な役目をもっているんだね
始末書
出せー



ところで今までは何気なく「セットされる」っていってたけど正確にいい直すと
なんだ今までテキトーに言ってたのか
セット リセット
フラグ
(C, Z, V, N)
各フラグは条件が成立すればセットされしなければリセットされるの
条件成立 条件不成立
"1" セット
"0" リセット
つまり条件が不成立ならフラグが0にリセットされるという事なの
むんず!
K子さんK子さん質問があるんだけど――
あのねフラグってCPUのどこにあるの?
フラグ
N Z C V
あっごめんごめん説明があとになっちゃったけどフラグはCCR(コンディション・コード・レジスタ)の中にあるのよ
CCR(コンディション・コード・レジスタ)
E F H I N Z V C
E:割込みエントリ・フラグ
F:FIRQマスク ・フラグ
H:ハーフキャリー・フラグ
I:IRQマスク ・フラグ
N:ネガティブ ・フラグ
Z:ゼロ ・フラグ
V:オーバーフロー・フラグ
C:キャリー/ボロー・フラグ
フラグの種類ってまだまだあるんだね





LDA　5000番地
LDB　5001番地　←──Ⓝフラグ□[1]
ADDD 5002番地　←──Ⓒフラグ□[2]
STD　5000番地
LDD　5000番地　←──Ⓩフラグ□[3]
END

5000番地　$88
5001番地　$88
5002番地　$77
5003番地　$78

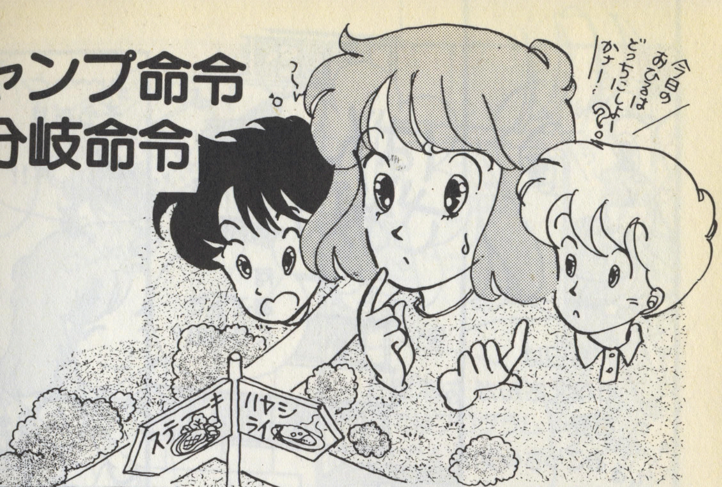




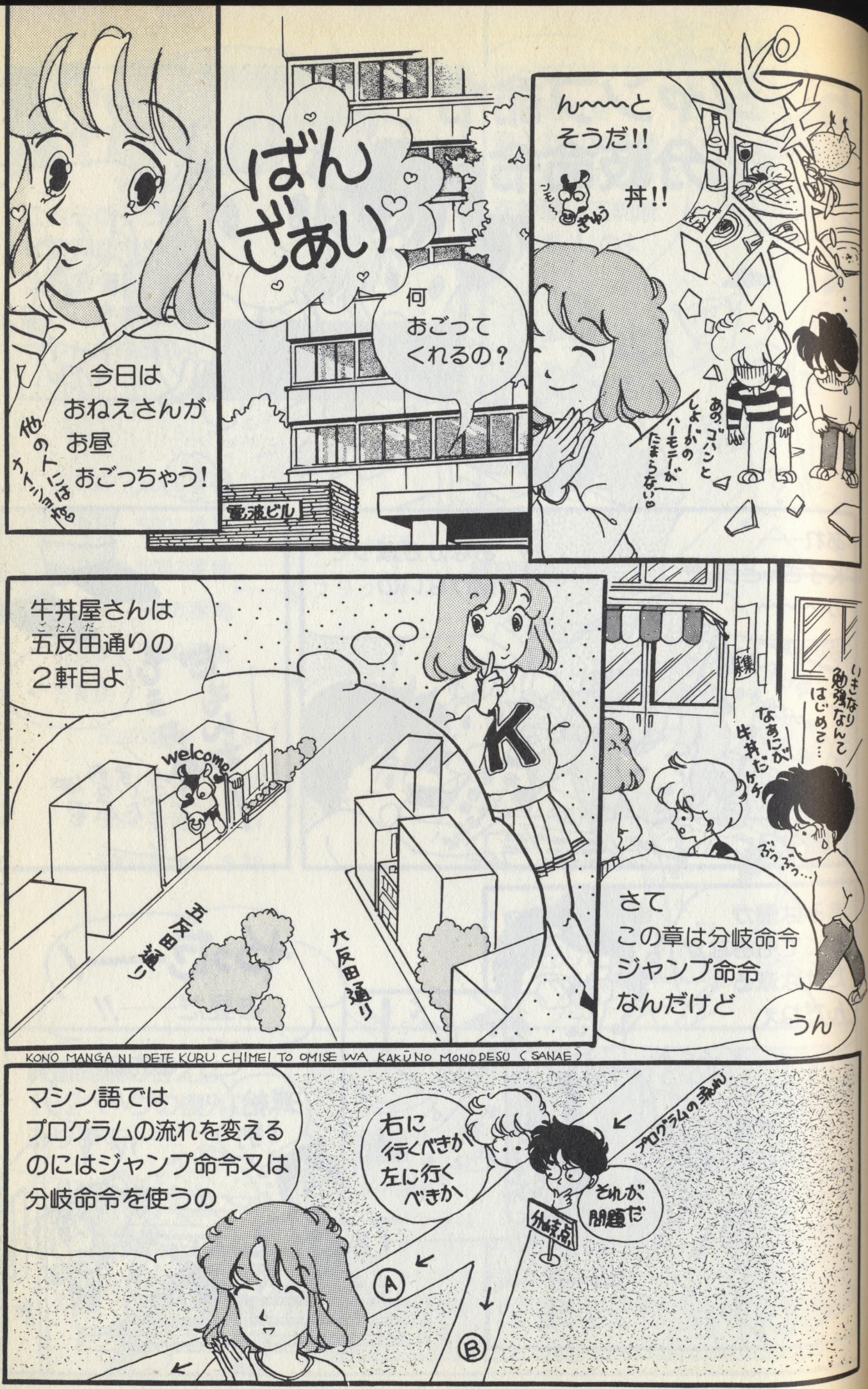
コーヒーブレーク「6809とZ80」

モトローラ社の6809，ザイログ社のZ80，なにかと比較される８ビットの代表的なＣＰＵですが，両者の違いはなんでしょうか？一言でいえば，ポストバイトという当時（1974年発表）としては斬新なハードウエアを採用したアドレッシング指定形式にあります。メモリ操作が貧弱なZ80に比べてスタック操作も強力な6809は高級言語指向を意識したＣＰＵであるといえます。

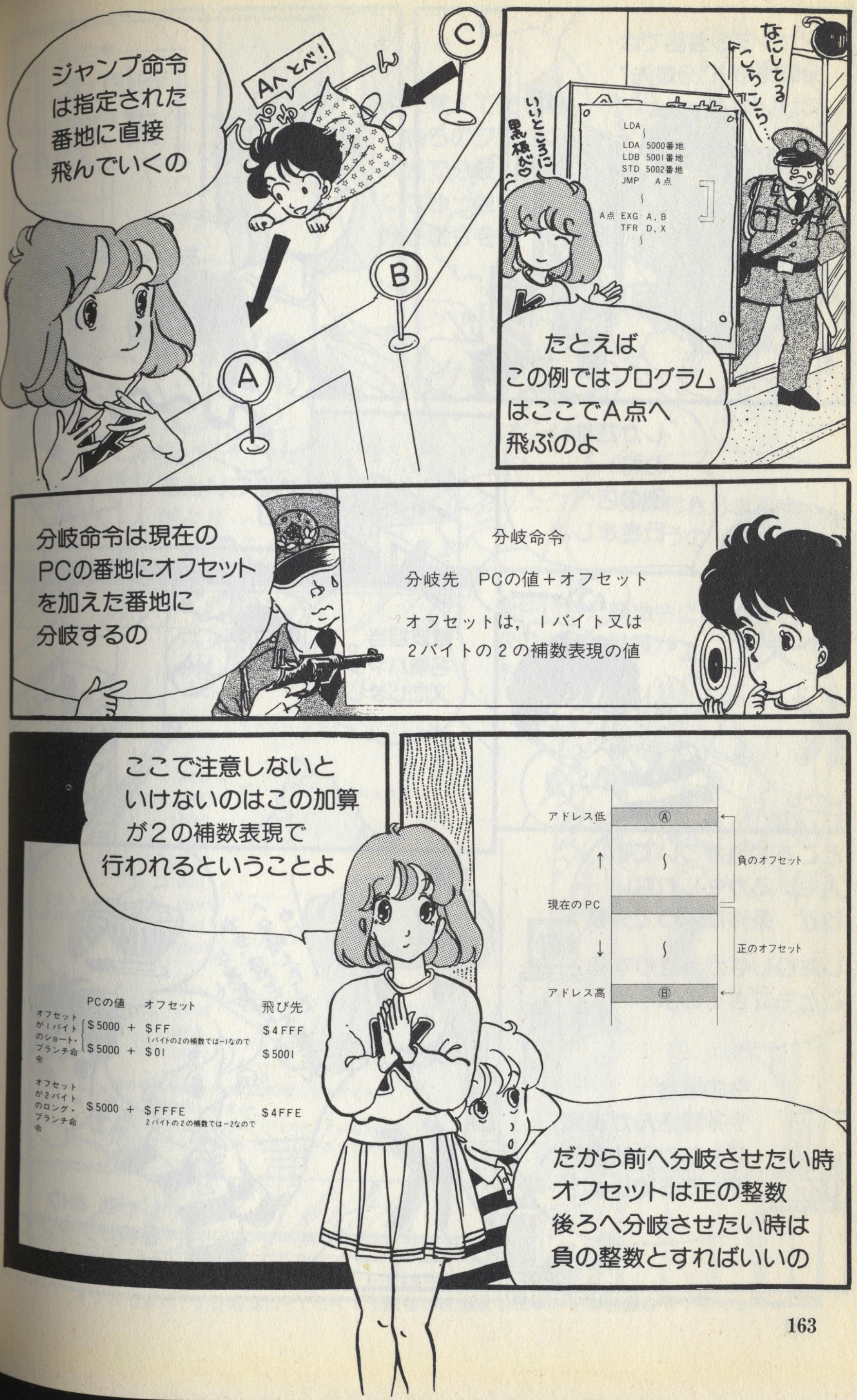
3-5 ジャンプ命令 分岐命令







●160ページの答え● N=1, C=1, Z=1



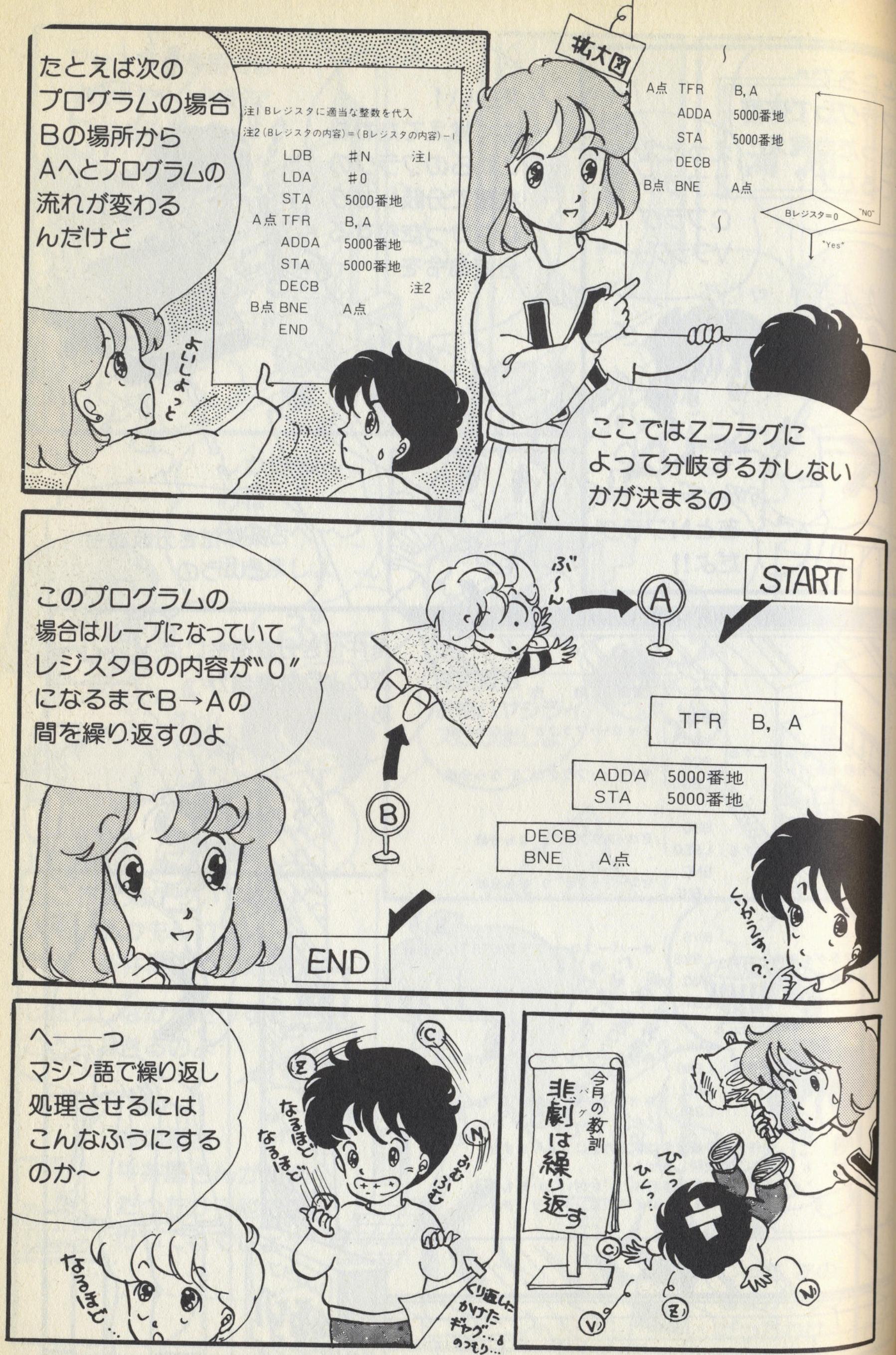


＊アセンブラが自動的にラベル又は分岐先の番地をオフセットに変換してくれるからです。

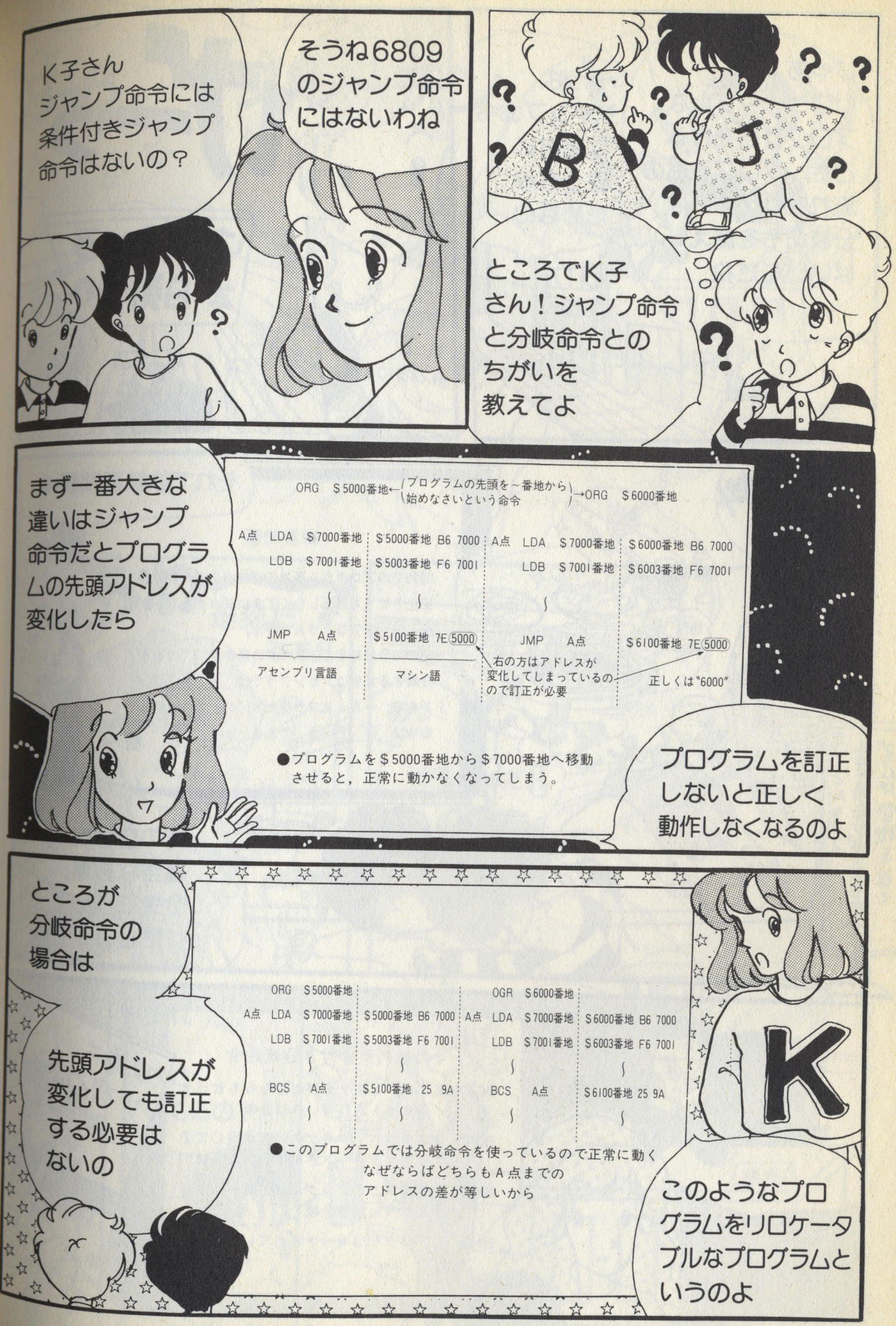


	アセンブリ言語	動　作
Cフラグを条件とする分岐命令	BCS LBCS	キャリー・フラグが"1"なら分岐
	BCC LBCC	キャリー・フラグが"0"なら分岐
Zフラグを条件とする分岐命令	BEQ LBEQ	ゼロ・フラグが"1"なら分岐
	BNE LBNE	ゼロ・フラグが"0"なら分岐
Vフラグを条件とする分岐命令	BVS LBVS	オーバーフロー・フラグが"1"なら分岐
	BVC LBVC	オーバーフロー・フラグが"0"なら分岐
Nフラグを条件とする分岐命令	BMI LBMI	ネガティブ・フラグが"1"なら分岐
	BPL LBPL	ネガティブ・フラグが"0"なら分岐

条件付き分岐命令は以上の他にもありますが
ここでは説明しません
アセンブリ言語の前に"L"が付いているもの
はオフセットが2バイトのロングブランチ命
令です



たとえば次の
プログラムの場合
Bの場所から
Aへとプログラムの
流れが変わる
んだけど
注1 Bレジスタに適当な整数を代入
注2 (Bレジスタの内容)＝(Bレジスタの内容)－1
LDB #N 注1
LDA #0
STA 5000番地
A点 TFR B, A
ADDA 5000番地
STA 5000番地
DECB 注2
B点 BNE A点
END
拡大図
A点 TFR B, A
ADDA 5000番地
STA 5000番地
DECB
B点 BNE A点
Bレジスタ＝0
"NO"
"Yes"
ここではZフラグに
よって分岐するかしない
かが決まるの
このプログラムの
場合はループになっていて
レジスタBの内容が"0"
になるまでB→Aの
間を繰り返すのよ
START
TFR B, A
ADDA 5000番地
STA 5000番地
DECB
BNE A点
END
へ――っ
マシン語で繰り返し
処理させるには
こんなふうにする
のか～
今月の教訓
悲劇は繰り返す



K子さん
ジャンプ命令には
条件付きジャンプ
命令はないの？
そうね6809
のジャンプ命令
にはないわね
ところでK子
さん！ジャンプ命令
と分岐命令との
ちがいを
教えてよ
まず一番大きな
違いはジャンプ
命令だとプログラ
ムの先頭アドレスが
変化したら
ORG ＄5000番地←(プログラムの先頭を～番地から始めなさいという命令)→ORG ＄6000番地
A点 LDA ＄7000番地 ＄5000番地 B6 7000 A点 LDA ＄7000番地 ＄6000番地 B6 7000
LDB ＄7001番地 ＄5003番地 F6 7001 LDB ＄7001番地 ＄6003番地 F6 7001
JMP A点 ＄5100番地 7E 5000 JMP A点 ＄6100番地 7E 5000
アセンブリ言語
マシン語
右の方はアドレスが変化してしまっているので訂正が必要
正しくは"6000"
●プログラムを＄5000番地から＄7000番地へ移動させると，正常に動かなくなってしまう。
プログラムを訂正
しないと正しく
動作しなくなるのよ
ところが
分岐命令の
場合は
先頭アドレスが
変化しても訂正
する必要は
ないの
ORG ＄5000番地 OGR ＄6000番地
A点 LDA ＄7000番地 ＄5000番地 B6 7000 A点 LDA ＄7000番地 ＄6000番地 B6 7000
LDB ＄7001番地 ＄5003番地 F6 7001 LDB ＄7001番地 ＄6003番地 F6 7001
BCS A点 ＄5100番地 25 9A BCS A点 ＄6100番地 25 9A
●このプログラムでは分岐命令を使っているので正常に動く
なぜならばどちらもA点までの
アドレスの差が等しいから
K
このようなプロ
グラムをリロケータ
ブルなプログラムと
いうのよ

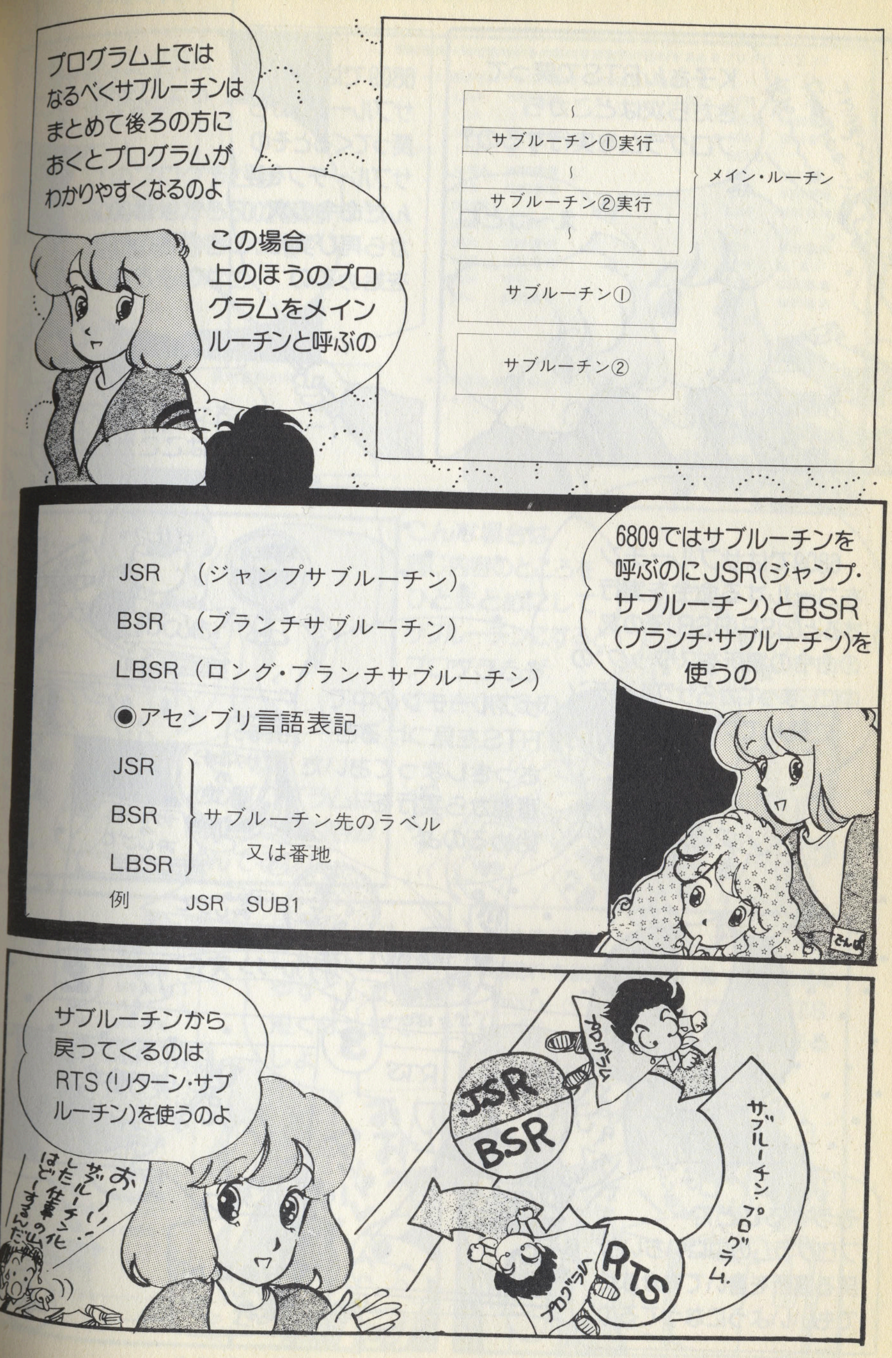

じゃあ
リロケータブルな
プログラムを作る
とき　プログラムの
流れをかえるには
分岐命令を使え
ばいいんだね
そうよ
ジャンプ命令
は絶対に
使っちゃ
だめなの
げーっ
はやしらいす
本日売切れ
さて，
やっと
おひるに
….
今月の給料
ナシ….
ごはん
きょーくん　その2.
いきあたりばったりのプロ
グラムは空腹を召く
それでは練習問題よ
練習問題
6809でのプログラム手法でプログラムの番地を
変化させても訂正しなくて良いプログラミング
を ① なプログラムと呼ぶ。
6809の分岐命令でZフラグの条件が"1"のとき
分岐する命令はアセンブラでは ② と ③
である。一方，Vフラグが"0"のとき分岐する
命令は ④ と ⑤ である。
次の空白を
適当な言葉，
命令でうめて
ください
答えは
170ページの
欄外です。
その他の条件付き分岐命令
BGE　NフラグとVフラグがどちらもセット／リセットされていれば分岐
BGT　NフラグとVフラグの内容が同じでZフラグがリセットされていれば分岐
BHI　CフラグとZフラグがどちらもリセットされていれば分岐
BHS　Cフラグがリセットされていれば分岐
次章は，
サブルーチン
の勉強
でーす

3-6 サブルーチン


こんど
お給料を
あげてあげようっと
K子はよく働く
いい子だねぇ
ゴロゴロ
うわ〜！
遅刻だ
ああ!!
電波ビル
どどどどど
あっ！
K子さんだ!!
あれ
なにしてるの？
立たされ
てんのよ
バカぁ！
私は
情けない…
組長
K子さん
こんにちは!!
バターン!!



●168ページの答え●
①リロケータブル, ②BEQ, ③LBEQ, ④BVC, ⑤LBVC

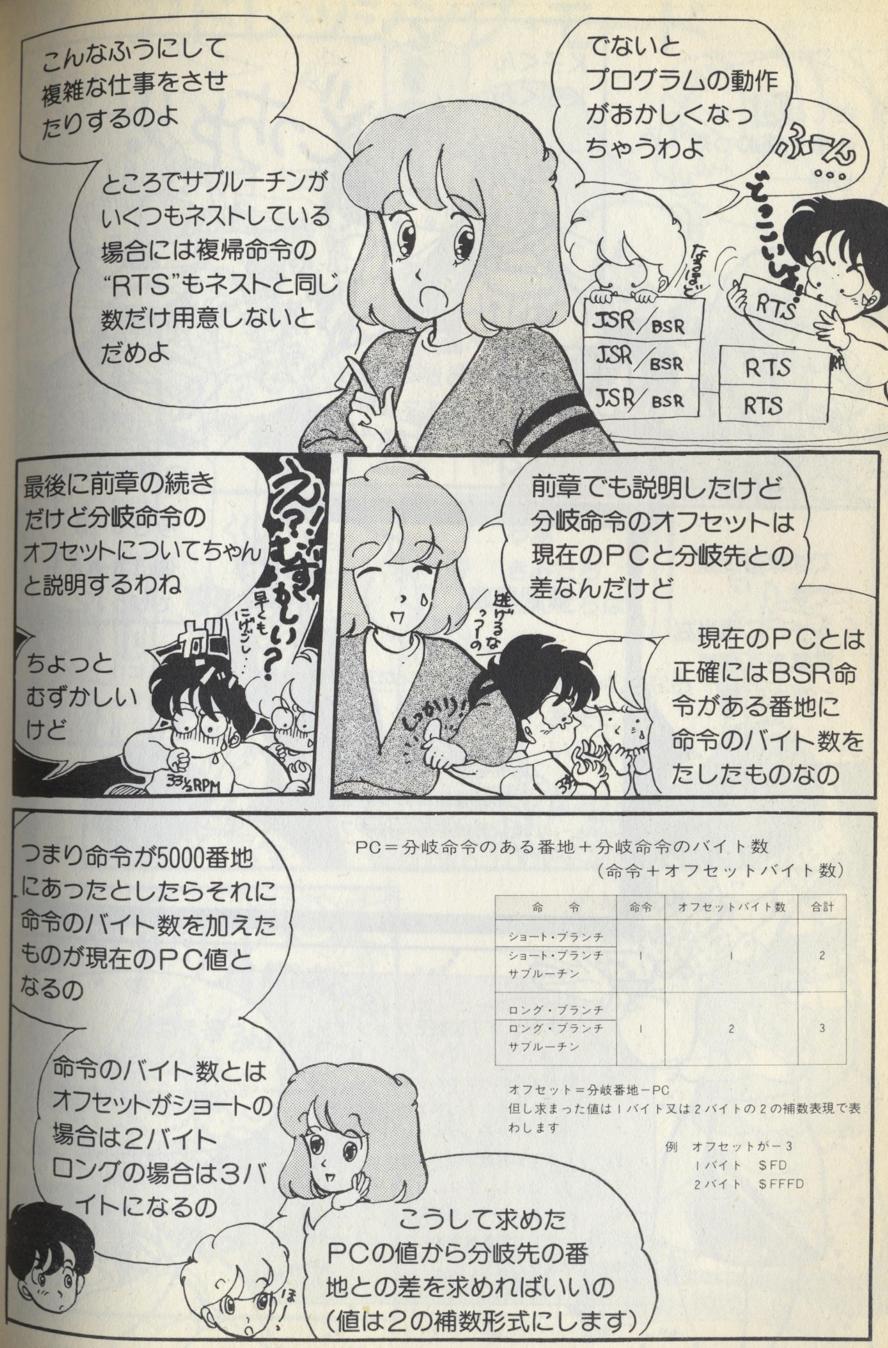




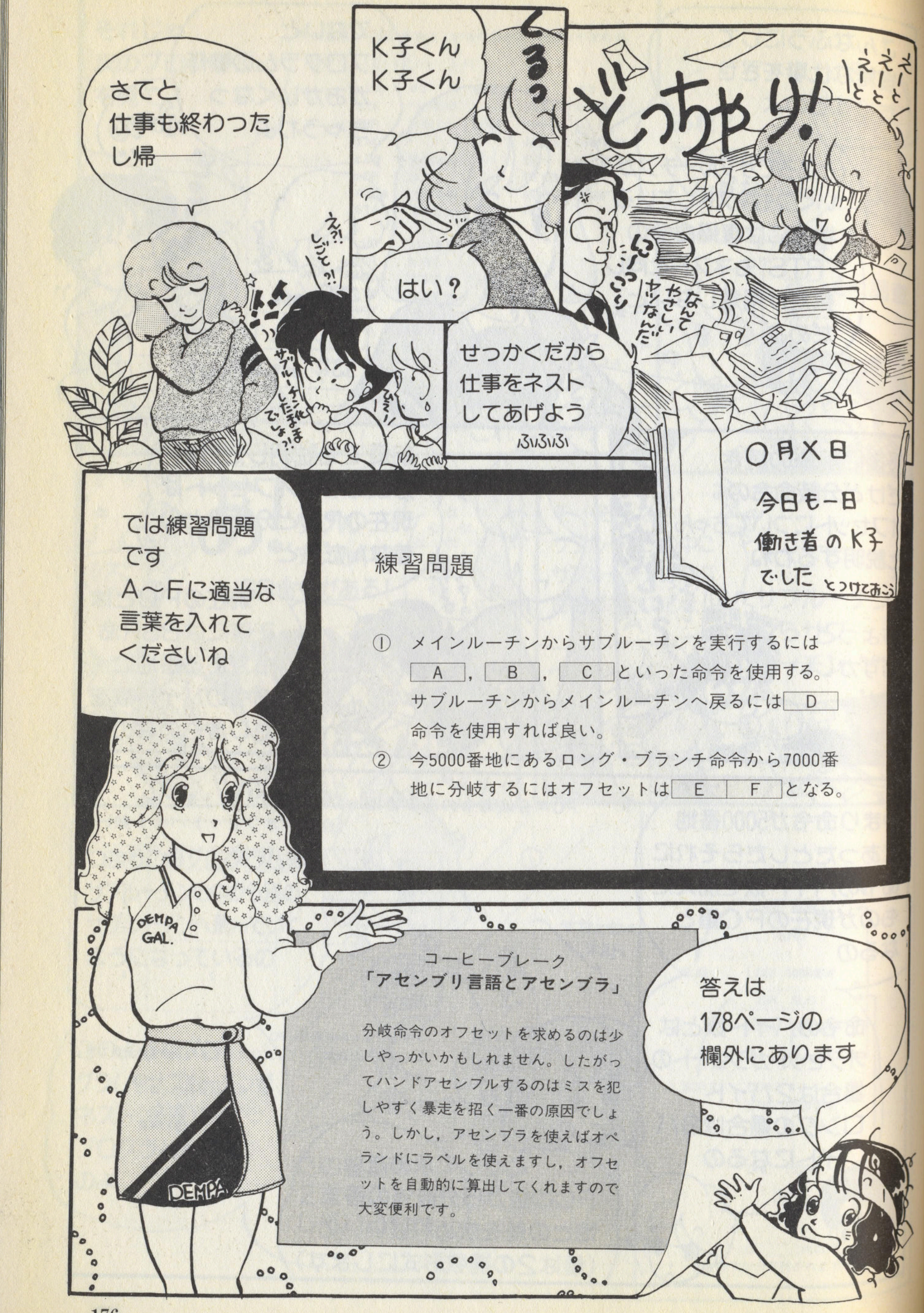
＊具体的にはスタックポインタで表わされるメモリ番地にセーブします







命令	命令	オフセットバイト数	合計
ショート・ブランチ ショート・ブランチサブルーチン	1	1	2
ロング・ブランチ ロング・ブランチサブルーチン	1	2	3



練習問題

① メインルーチンからサブルーチンを実行するには [A], [B], [C] といった命令を使用する。サブルーチンからメインルーチンへ戻るには [D] 命令を使用すれば良い。

② 今5000番地にあるロング・ブランチ命令から7000番地に分岐するにはオフセットは [E] [F] となる。

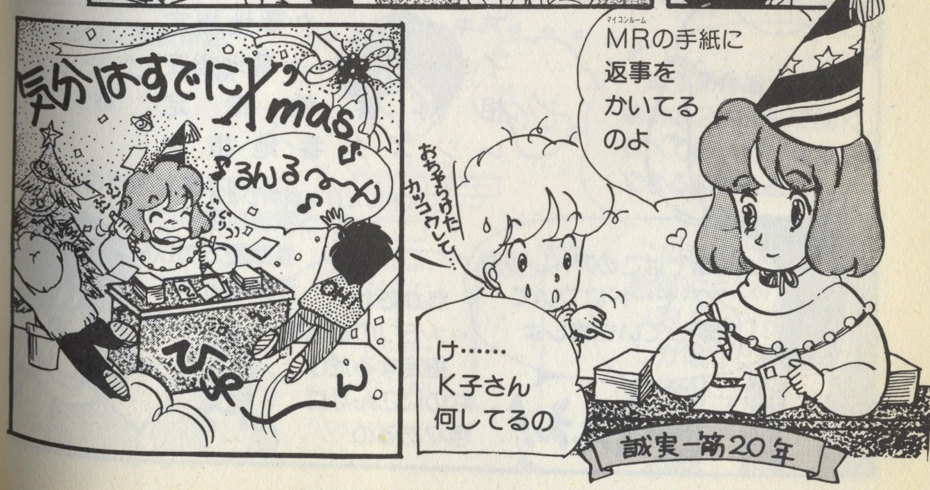
コーヒーブレーク

「アセンブリ言語とアセンブラ」

分岐命令のオフセットを求めるのは少しやっかいかもしれません。したがってハンドアセンブルするのはミスを犯しやすく暴走を招く一番の原因でしょう。しかし，アセンブラを使えばオペランドにラベルを使えますし，オフセットを自動的に算出してくれますので大変便利です。

3-7 アドレッシング・モード



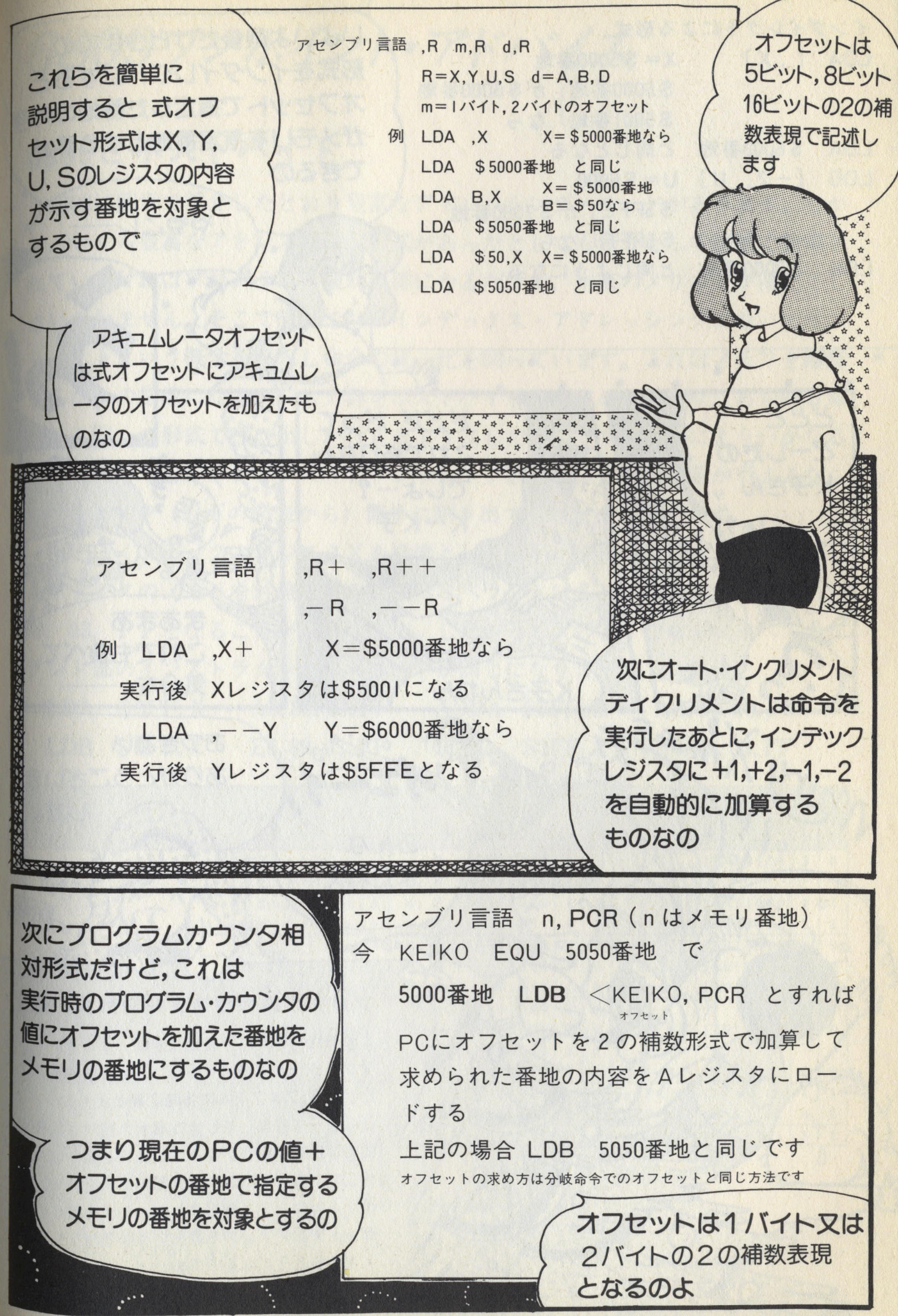






●176ページの答え●
A＝JSR, B＝BSR, C＝LBSR, D＝RTS, E＝07, F＝CD


でも――
それじゃあまり
使い道がない
んじゃあ
No No.
そんなこと
ないの
つまり命令とオフ
セットあわせて2バイト
だから拡張番地指定
より処理が
速いの
でも256
バイト分
のメモリしか
指定できない
んなら……
拡張番地指定命令 + オフセット = 3バイト
1バイト + 2バイト
直接番地指定命令 + オフセット = 2バイト
1バイト + 1バイト
2バイト
1バイト
DP（ダイレクトページレジスタ）
オフセット
DP オフセット
DPレジスタを上位バイトに設定し，下位バイトにオフセットを設定する。
6809の直接
番地指定では
このDPレジスタと
オフセットを合わ
せたものを番地
とするの
そこで6809の
DPレジスタの
登場となるの
なるほど
こうすれば
64Kのメモリを
すべて指定
できるのか
わかった？
でももちろんDP
レジスタに値が
セットされていれば
話だけど
LDA ＃$80
TFR A, DP
LDA <$80 ← LDA $8080と同じ
オフセットを1バイトにするという記号
$F080番地をアクセスしたい場合は，最初のプログラムをLDA ＃$F0とすればいい
じゃ例題ね
仲間ハズレ
わあー便利!!
1バイトのオフセットで64K
すべてのメモリを指定できる
なんて
じゃあ最後に
インデックス・アド
レッシングについて
ね
この方法には
五つの形式が
あるの
インデックス番地指定
○式オフセット形式
○アキュムレータオフセット形式
○自動インクリメント形式
○自動デクリメント形式
○プログラムカウンタ相対形式


これらを簡単に
説明すると 式オフ
セット形式はX, Y,
U, Sのレジスタの内容
が示す番地を対象と
するもので
アセンブリ言語 ,R m,R d,R
R=X,Y,U,S d=A,B,D
m=1バイト，2バイトのオフセット
例 LDA ,X X=$5000番地なら
LDA $5000番地 と同じ
LDA B,X X=$5000番地 B=$50なら
LDA $5050番地 と同じ
LDA $50,X X=$5000番地なら
LDA $5050番地 と同じ
オフセットは
5ビット，8ビット
16ビットの2の補
数表現で記述し
ます
アキュムレータオフセット
は式オフセットにアキュムレ
ータのオフセットを加えたも
のなの
アセンブリ言語 ,R+ ,R++
,－R ,－－R
例 LDA ,X+ X=$5000番地なら
実行後 Xレジスタは$5001になる
LDA ,－－Y Y=$6000番地なら
実行後 Yレジスタは$5FFEとなる
次にオート・インクリメント
ディクリメントは命令を
実行したあとに，インデック
レジスタに+1,+2,－1,－2
を自動的に加算する
ものなの
次にプログラムカウンタ相
対形式だけど，これは
実行時のプログラム・カウンタの
値にオフセットを加えた番地を
メモリの番地にするものなの
つまり現在のPCの値＋
オフセットの番地で指定する
メモリの番地を対象とするの
アセンブリ言語 n, PCR（nはメモリ番地）
今 KEIKO EQU 5050番地 で
5000番地 LDB <KEIKO, PCR とすれば
オフセット
PCにオフセットを2の補数形式で加算して
求められた番地の内容をAレジスタにロ
ードする
上記の場合 LDB 5050番地と同じです
オフセットの求め方は分岐命令でのオフセットと同じ方法です
オフセットは1バイト又は
2バイトの2の補数表現
となるのよ

インダイレクトによる形式

```
LDA 〔,X〕       X=$5000番地
                $5000番地 } が$6000番地
                $5001番地 } なら
LDA  $6000番地  と同じとなる
LDB 〔−2,U〕    U=$6000
                $5FFE } が$7000番地
                $5FFF } なら
LDB  $7000番地  と同じようになる
```



プログラムカウンタ相対形式の補足

この形式でのメモリ番地指定は，リロケータブルなプログラムを記述する時に必要となるものです。たとえば，プログラムの番地変換とともにデータもともなって移動してしまう場合，絶対番地指定形式では正しく動作させることはできなくなります。そこでプログラムカウンタ相対形式を使い，プログラムカウンタと指定されたメモリ番地とのオフセットで実アドレスを算出すれば問題なくアクセスできるようになるのです。

6809ワンポイント・アドバイス

6809とポストバイト

6809は前章で紹介したとおり豊富なアドレッシング・モードがあります。しかし，いくら豊富なアドレスの指定形式があったとしても命令として複雑で整備されていなければプログラム作業が複雑になるだけで，本来のメリットが薄れてしまいかねません。そこで6809では，インデックス・アドレッシングにおいてポストバイトという概念を導入し命令の統一化を図っています。これはメモリを修飾するための指標になるレジスタ群をポストバイト化することにより，あらゆる命令から同一の形式で呼び出しが行えるようにしたものです。これによりいかなる命令であってもインデックス・アドレッシング形式でメモリの指定が行えるならオペランドを（以下の図表から）簡単に割り出すことができるのです。

例えばインデックスレジスタXを指標とし$3FFバイトのオフセットを加算したアドレスのメモリを対照とする命令（オペランドは$3FF，X）は，図表から89，03，FFとなることがわかります。したがって次のような命令はすべてオペランド部のインストラクション・コードが同じになります。

```
LDA  $3FF,X   A6,89,03,FF   ADDA  $3FF,X   AB,89,03,FF
LDB  $3FF,X   E6,89,03,FF   SUBA  $3FF,X   A0,89,03,FF
```

アドレッシング・モード			インダイレクトでない場合				インダイレクトの場合			
			アセンブラ形式	ポストバイト	+~	+#	アセンブラ形式	ポストバイト	+~	+#
インデックス	式オフセット	オフセットなし	,R	1RR00100	0	0	〔,R〕	1RR10100	3	0
		5ビット・オフセット	n,R	0RRnnnnn	1	0	—	—	—	—
		8ビット・オフセット	n,R	1RR01000	1	1	〔n,R〕	1RR11000	4	1
		16ビット・オフセット	n,R	1RR01001	4	2	〔n,R〕	1RR11001	7	2
	アキュムレータオフセット	Aレジスタ・オフセット	A,R	1RR00110	1	0	〔A,R〕	1RR10110	4	0
		Bレジスタ・オフセット	B,R	1RR00101	1	0	〔B,R〕	1RR10101	4	0
		Dレジスタ・オフセット	D,R	1RR01011	4	0	〔D,R〕	1RR11011	7	0
	自動インクリメント／デクリメント	インクリメント(+1)	,R+	1RR00000	2	0	—	—	—	—
		インクリメント(+2)	,R++	1RR00001	3	0	〔,R++〕	1RR10001	6	0
		デクリメント (−1)	,−R	1RR00010	2	0	—	—	—	—
		デクリメント (−2)	,−−R	1RR00011	3	0	〔,−−R〕	1RR10011	6	0
プログラムカウンタ相対		8ビット・オフセット	n,PCR	1××01100	1	1	〔n,PCR〕	1××11100	4	1
		16ビット・オフセット	n,PCR	1××01101	5	2	〔n,PCR〕	1××11101	8	2
拡張間接		16ビット・アドレス	—	—	—	—	〔n〕	10011111	5	2

〔記号の説明〕

R：X，Y，S，Uレジスタのうちの一つ。
RR：指定されたレジスタを表す。
Xレジスタの場合　00
Yレジスタの場合　01
Uレジスタの場合　10
Sレジスタの場合　11

X：任意
+~：追加されるマシンサイクル数
+#：追加されるバイト数

絵で見るマシン語プログラム
HOW TO マシン語
Z80/6809マスター編

発行日●昭和62年12月24日初版発行
●昭和63年3月25日第2刷発行

定　価●980円（〒250円）

著　者●絵/秋山早苗・監修/大沢正道

発行人●平山秀雄

発行所●㈱電波新聞社
〒141 東京都品川区東五反田1-11-15
電話03(445)6111　振替 東京 5-51961

印刷所●大日本印刷㈱

製本所●㈱堅省堂

ISBN4-88554-153-X C3055 ¥980E